МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Российский государственный гуманитарный университет»
(ФГАОУ ВО «РГГУ»)

ИНСТИТУТ ИНФОРМАЦИОННЫХ НАУК И ТЕХНОЛОГИЙ БЕЗОПАСНОСТИ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ СИСТЕМ И БЕЗОПАСНОСТИ
Кафедра информационных технологий и систем

ПРОГРАММНАЯ ИНЖЕНЕРИЯ

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

09.03.03 Прикладная информатика
Код и наименование направления подготовки

Прикладной искусственный интеллект
Наименование направленности (профиля)

Уровень высшего образования: *бакалавриат*

Форма обучения: *очная*

РПД адаптирована для лиц
с ограниченными возможностями
здоровья и инвалидов

Москва 2026

ПРОГРАММНАЯ ИНЖЕНЕРИЯ
Рабочая программа дисциплины

Составитель(и):
старший преподаватель кафедры информационных технологий и систем Е.П. Охапкина

УТВЕРЖДЕНО
Протокол заседания кафедры ИТС
№ 5 от 11.12.2025

ОГЛАВЛЕНИЕ

1. Пояснительная записка

1.1. Цель и задачи дисциплины

Цель дисциплины: приобретение знаний, навыков и умений в области применения современных подходов к проектированию, разработке, тестированию и эксплуатации программных продуктов.

Задачи:

1. изучение и сравнительный анализ современных процессов проектирования и разработки программных продуктов;
2. изучение принципов и методов оценки качества и управления качеством программного продукта;
3. приобретение практических навыков формирования и анализа требований, оценки качества и тестирования программных продуктов.

1.2. Перечень планируемых результатов обучения по дисциплине, соотнесенных с индикаторами достижения компетенций

Компетенция	Индикаторы компетенций	Результаты обучения
ОПК-2 - Способен понимать принципы работы современных информационных технологий и программных средств, в том числе отечественного производства, и использовать их при решении задач профессиональной деятельности	ОПК-2.1 - Понимает принципы работы современных информационных технологий и программных средств, в том числе отечественного производства.	Знать: современные процессы проектирования и разработки программных продуктов.
	ОПК-2.2 - Обоснованно выбирает современные информационные технологии и программные средства, в том числе отечественного производства для решении задач профессиональной деятельности.	Уметь: проводить сравнительный анализ процессов проектирования и разработки программных продуктов и делать обоснованный выбор.
	ОПК – 2.3 - Использует современные информационные технологии и	Владеть: информацией о процессах разработки и жизненном цикле программного обеспечения.

	программные средства, в том числе отечественного производства, при решении задач профессиональной деятельности.	
ОПК-4 Способен участвовать в разработке стандартов, норм и правил, а также технической документации, связанной с профессиональной деятельностью	ОПК-4.1 Знает основные стандарты оформления технической документации на различных стадиях жизненного цикла информационной системы	Знать: принципы управления качеством программного обеспечения
	ОПК-4.2 Умеет применять стандарты оформления технической документации на различных стадиях жизненного цикла информационной системы	Уметь: выполнять формирование и анализ требований для разработки программных продуктов.
	ОПК-4.3 Владеет навыками составления технической документации на различных этапах жизненного цикла информационной системы	Владеть: инструментарием для тестирования программных продуктов.
ОПК-5 Способен инсталлировать программное и аппаратное обеспечение для информационных и автоматизированных систем	ОПК-5.1 Знает основы системного администрирования, администрирования СУБД, современные стандарты информационного взаимодействия систем	Знать: методы тестирования программного продукта.
	ОПК-5.2 Умеет выполнять параметрическую настройку информационных и автоматизированных систем	Уметь: разрабатывать документацию, необходимую для тестирования программного продукта; выполнять тестирование программного

		продукта.
	ОПК-5.3 Владеет навыками инсталляции программного и аппаратного обеспечения информационных и автоматизированных систем	Владеть: инструментарием для разработки программных продуктов.

1.3. Место дисциплины в структуре образовательной программы

Дисциплина «Программная инженерия» относится к обязательной части блока дисциплин учебного плана.

Для освоения дисциплины необходимы знания, умения и владения, сформированные в ходе изучения дисциплин Алгоритмы и структуры данных, Математические модели и методы, Базы данных, Программирование.

В результате освоения дисциплины формируются знания, умения и владения, необходимые для изучения следующих дисциплин и прохождения практик: Моделирование бизнес процессов и проектирование систем, Управление информационными системами, Управление ИТ проектами.

2. Структура дисциплины

Общая трудоёмкость дисциплины составляет 4 з.е., 144 академических часа.

Структура дисциплины для очной формы обучения

Объем дисциплины в форме контактной работы обучающихся с педагогическими работниками и (или) лицами, привлекаемыми к реализации образовательной программы на иных условиях, при проведении учебных занятий:

Семестр	Тип учебных занятий	Количество часов
5	Лекции	8
5	Практические занятия	20
	Всего:	28

Семестр	Тип учебных занятий	Количество часов
6	Лекции	8
6	Практические занятия	20
	Всего:	28

Объем дисциплины (модуля) в форме самостоятельной работы обучающихся составляет 44 академических часа в пятом семестре и 26 академических часа в шестом семестре.

3. Содержание дисциплины

№	Наименование	Содержание

	раздела дисциплины	
1.	Тема 1. Организация процесса разработки ПО	Основные понятия. Классификация процессов программной инженерии. Базис процессов разработки ПО. Стратегии разработки ПО.
2.	Тема 2. Руководство программным проектом	Основные понятия руководства программным проектом. Планирование программного проекта. Контроль программного проекта. Управление рисками. Управление персоналом. Управление документацией. Управление конфигурацией.
3.	Тема 3. Оценка при планировании программного проекта	Метрики оценки проекта. Выполнение оценки в ходе планирования проекта. Предварительная оценка программного проекта. Анализ чувствительности программного проекта.
4.	Тема 4. Формирование и анализ требований. Знакомство с языком UML	Виды требований к ПО. Формирование требований. Анализ требований. Объекты. Классы. Базис языка визуального моделирования .
5.	Тема 5. Классические методы анализа	Структурный анализ. Методы анализа, ориентированные на структуры данных. Метод анализа Джексона.
6.	Тема 6. Основы проектирования программных систем	Особенности процесса синтеза программных систем. Особенности архитектурного этапа проектирования Структурирование системы. Моделирование управления. Декомпозиция подсистем. Связность модуля. Сложность программной системы.
7.	Тема 7. Классические методы проектирования	Метод структурного проектирования. Метод проектирования Джексона.
8.	Тема 8. Основы объектно-ориентированного представления программных систем. Объектно-ориентированная разработка требований. Автоматизированная разработка визуальных моделей	Абстрагирование. Инкапсуляция. Модульность. Иерархическая организация. Объекты. Классы. Базис языка визуального моделирования (продолжение). Формирование требований с помощью диаграммы Use Case, диаграммы деятельности. Анализ требований с помощью диаграмм взаимодействия. Моделирование с помощью диаграмм конечных автоматов. Общая характеристика и создание моделей программной системы с помощью CASE-средств, поддерживающих язык UML.
9.	Тема 9. Объектно-ориентированное проектирование и реализация. Организация тестирования ПО	Архитектурное проектирование. Детальное проектирование. Принцип построения паттернов. Проектирование пользовательского интерфейса. Основы компонентной объектной модели. Методика тестирования программных систем. Тестирование элементов. Тестирование интеграции. Тестирование правильности. Системное тестирование. Искусство отладки.

10.	Тема 10. Обеспечение качества программных систем.	Определение качества ПО. Факторы качества. Деятельность по обеспечению качества. Технические проверки и аудиты. Инспектирование. Верификация и валидация. План обеспечения качества ПО.

4. Образовательные технологии

Для проведения учебных занятий по дисциплине используются различные образовательные технологии. Для организации учебного процесса может быть использовано электронное обучение и (или) дистанционные образовательные технологии.

5. Оценка планируемых результатов обучения

5.1 Система оценивания

Семестр 1:

Форма контроля	**Макс. количество баллов**	
	За одну работу	**Всего**
Текущий контроль: Прием докладов Практическая работа № 1, защита отчета Практическая работа № 2, защита отчета Практическая работа № 3, защита отчета Практическая работа № 4, защита отчета	 10 баллов 10 баллов 10 баллов 20 баллов 10 баллов	 10 баллов 10 баллов 10 баллов 20 баллов 10 баллов
Промежуточная аттестация *зачет с оценкой*		40 баллов
Итого за семестр		100 баллов

Семестр 2:

Форма контроля	**Макс. количество баллов**	
	За одну работу	**Всего**
Текущий контроль: Практическая работа № 5, защита отчета Практическая работа № 6, защита отчета Практическая работа № 7, защита отчета	 20 баллов 20 баллов 20 баллов	 20 баллов 20 баллов 20 баллов
Промежуточная аттестация *экзамен*		40 баллов
Итого за семестр		100 баллов

Полученный совокупный результат конвертируется в традиционную шкалу оценок и в шкалу оценок Европейской системы переноса и накопления кредитов (European Credit Transfer System; далее – ECTS) в соответствии с таблицей:

100-балльная шкала	Традиционная шкала		Шкала ECTS
95 – 100	отлично		A
83 – 94		зачтено	B

68 – 82	хорошо		C
56 – 67	удовлетворительно		D
50 – 55			E
20 – 49	неудовлетворительно	не зачтено	FX
0 – 19			F

5.2 Критерии выставления оценки по дисциплине

Баллы/ Шкала ECTS	**Оценка по дисциплине**	**Критерии оценки результатов обучения по дисциплине**
100-83/ A,B	отлично/ зачтено	Выставляется обучающемуся, если он глубоко и прочно усвоил теоретический и практический материал, может продемонстрировать это на занятиях и в ходе промежуточной аттестации. Обучающийся исчерпывающе и логически стройно излагает учебный материал, умеет увязывать теорию с практикой, справляется с решением задач профессиональной направленности высокого уровня сложности, правильно обосновывает принятые решения. Свободно ориентируется в учебной и профессиональной литературе. Оценка по дисциплине выставляются обучающемуся с учётом результатов текущей и промежуточной аттестации. Компетенции, закреплённые за дисциплиной, сформированы на уровне – «высокий».
82-68/ C	хорошо/ зачтено	Выставляется обучающемуся, если он знает теоретический и практический материал, грамотно и по существу излагает его на занятиях и в ходе промежуточной аттестации, не допуская существенных неточностей. Обучающийся правильно применяет теоретические положения при решении практических задач профессиональной направленности разного уровня сложности, владеет необходимыми для этого навыками и приёмами. Достаточно хорошо ориентируется в учебной и профессиональной литературе. Оценка по дисциплине выставляются обучающемуся с учётом результатов текущей и промежуточной аттестации. Компетенции, закреплённые за дисциплиной, сформированы на уровне – «хороший».
67-50/ D,E	удовлетво-рительно/ зачтено	Выставляется обучающемуся, если он знает на базовом уровне теоретический и практический материал, допускает отдельные ошибки при его изложении на занятиях и в ходе промежуточной аттестации. Обучающийся испытывает определённые затруднения в применении теоретических положений при решении практических задач профессиональной направленности стандартного уровня сложности, владеет необходимыми для этого базовыми навыками и приёмами. Демонстрирует достаточный уровень знания учебной литературы по дисциплине. Оценка по дисциплине выставляются обучающемуся с учётом результатов текущей и промежуточной аттестации. Компетенции, закреплённые за дисциплиной, сформированы на уровне – «достаточный».
49-0/ F,FX	неудовлет-ворительно/ не зачтено	Выставляется обучающемуся, если он не знает на базовом уровне теоретический и практический материал, допускает грубые ошибки при его изложении на занятиях и в ходе промежуточной аттестации. Обучающийся испытывает серьёзные затруднения в применении теоретических положений при решении практических задач профессиональной направленности стандартного уровня сложности, не владеет необходимыми для этого навыками и приёмами. Демонстрирует фрагментарные знания учебной литературы по дисциплине. Оценка по дисциплине выставляются обучающемуся с учётом результатов текущей и промежуточной аттестации. Компетенции на уровне «достаточный», закреплённые за дисциплиной, не сформированы.

5.3 Оценочные средства (материалы) для текущего контроля успеваемости, промежуточной аттестации обучающихся по дисциплине

Контрольные вопросы зачета

1. Основные понятия программной инженерии
2. Официальная классификация процессов программной инженерии
3. Процессы соглашения
4. Процессы организационного обеспечения проекта
5. Процессы проекта
6. Технические процессы
7. Базис процессов разработки ПО
8. Модель «классический жизненный цикл»
9. Макетирование
10. Стратегии разработки ПО
11. Инкрементная модель
12. Спиральная модель
13. Компонентно-ориентированная модель
14. Тяжеловесные и облегченные процессы
15. Манифест гибкой разработки программного обеспечения
16. XP-процесс
17. Бережливая разработка программного обеспечения
18. Модели качества процессов разработки
19. Основные понятия руководства проектом
20. Начало проекта
21. Измерения меры и метрики
22. Процесс оценки
23. Анализ риска
24. Планирование
25. Трассировка и контроль
26. Планирование программного проекта
27. Размерно-ориентированные метрики
28. Функционально-ориентированные метрики
29. Выполнение оценки в ходе планирования проекта
30. Конструктивная модель стоимости
31. Модель композиции приложения
32. Модель раннего этапа проектирования
33. Предварительная оценка программного проекта
34. Анализ чувствительности программного проекта
35. Виды требований к программному обеспечению
36. Формирование требований
37. Анализ требований
38. Желаемые характеристики детального требования
39. Спецификация требований
40. Управление требованиями

Экзаменационный тест

001. Легкость применения программного обеспечения это:
-характеристики ПО, позволяющие минимизировать усилия пользователя по подготовке исходных данных, применению ПО
-отношение уровня услуг, предоставляемых ПО пользователю при заданных условиях, к объему используемых ресурсов
-характеристики ПО, позволяющие минимизировать усилия по внесению изменений для устранения в нем ошибок и по его модификации

002. Мобильность программного обеспечения это:
-способность ПО выполнять набор функций, которые удовлетворяют потребности пользователей
-способность ПС безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени
-способность ПО быть перенесенным из одной среды (аппаратной / программной) в другую

003. Укажите правильную последовательность этапов при каскадной модели жизненного цикла:
-Определение требований -> Тестирование -> Реализация
-Проектирование -> Реализация -> Тестирование
-Проектирование -> Определение требований -> Реализация

004. Устойчивость программного обеспечения — это:
-свойство, характеризующее способность ПС завершать автоматически корректное функционирование ПК, несмотря на неправильные (ошибочные) входные данные
-свойство, позволяющее противостоять преднамеренным или непреднамеренным деструктивным действиям пользователя
-свойство, характеризующее способность ПС продолжать корректное функционирование, несмотря на неправильные (ошибочные) входные данные.

005. UML — это:
-язык программирования, имеющий синтаксис схож с C --
-унифицированный язык визуального моделирования, использует нотацию диаграмм
-набор стандартов и спецификаций качества программного обеспечения.

006. При конструировании программного обеспечения процесс решения задачи составляет
-90—95%
-50%
-5—10%

007. При конструировании программного обеспечения на этапе разработки или выбора алгоритма решения реализуется следующее:
-архитектурная обработка программы
-выбор языка программирования
-совершенствование программы

008. Проектирование ПО в основном рассматривается как
-архитектурное проектирование
-коммуникационные методы
-детальные методы

009. На этапе тестирования пользователь выполняет следующее:
-синтаксическое отладки
-выбор тестов и метода тестирования
-определение формы выдачи результатов

010. Что из приведенного не является одним из методов проектирования программного обеспечения?
-структурное программирование
-объектно-ориентированное программирование
-алгебраическое программирование

011. Как называется процесс разбиения одной сложной задачи на несколько простых подзадач?
-абстракция
-декомпозиция
-реинжиниринг

012. Что из приведенного является критериями оценки удобства интерфейсов?
-скорость обучения
-адаптация к стилю работы пользователя
-все ответы правильные

013. Интерфейс пользователя — это
-набор методов взаимодействия компьютерной программы и пользователя этой программы
-набор методов для взаимодействия между программами
-способ взаимодействия между объектами

014. Интерфейс-это
-прежде всего, набор правил
-набор задач пользователя, которые он решает с помощью системы
-способ взаимодействия между объектами

015. Техническое задание — это
-документ объяснений для заказчика
-исходный документ для сдачи ПО в эксплуатацию
-выходной документ для проектирования, разработки автоматизированной системы

016. Анализ требований —
-отображение функций системы и ее ограничений в модели проблемы
-показатель, который определяет необходимые усилия для диагностики случаев отказов
-отображение частей программ, которые будут модифицироваться

017. Архитектура программной системы —
-декомпозиция решения для выделенного спектра задач домена на подсистемы или иерархию подсистем
-определение системы в терминах вычислительных составляющих (подсистем) и интерфейсов между ними, которое отражает правила декомпозиции проблемы на составляющие
-соответствующие вариации состава выделенных компонент.

018. Агрегация —
-отношения, утверждает наличие связи между понятиями, не уточняя зависимости их содержания и объемов;
-возможность для некоторого класса находиться одновременно в связи с одним элементом из определенного множества классов;
-объединение нескольких понятий в новое понятие, существенные признаки нового понятия при этом могут быть либо суммой компонент или существенно новыми (отношение «доля — целое»)

019. Ассоциация —
-возможность для некоторого класса находиться одновременно в связи с одним элементом из определенного множества классов
-объединение нескольких понятий в новее понятия, существенные признаки нового понятия о этом могут быть либо суммой компонент или существенно новыми (отношение «доля — целое»)

-самое общее отношение, утверждает наличие связи между понятиями, не уточняя зависимости их содержания и объемов

020. Валидация —
-обеспечение соответствия разработки требованиям ее заказчиков
-проверка правильности трансформации проекта в код реализации
-выявление всех ошибок

021. Верификация —
-обеспечение соответствия разработки требованиям ее заказчиков
-проверка правильности трансформации проекта в программу
-действия на каждой стадии жизненного цикла с проверки и подтверждения соответствия стандартам

022.Внешние метрики продукта:
-метрики надежности
-метрики размера
-метрики сложности

023. Внутренние метрики продукта:
-метрики сопровождения
-метрики годности
-метрики стиля

024. Продукты инженерии требований по методу С.Шлеер и С.Меллора:
-информационная модель системы
-описание интерфейсов сценариев и актеров
-неформальное описание сценариев и актеров

025. К процессу разработки ПО включает следующие процессы:
-сопровождения
-проектирование
-эксплуатация

026. Последовательность работ по каскадной моделью:
-требования, проектирование, реализация
-проектирование, сопровождение, тестирование
-требования, сопровождение, тестирование

027. Проектирование —
-преобразование требований в последовательность проектных решений по системе
-определение главных структурных особенностей системы
-определение подробностей функционирования и связей для всех компонент системы

028. Модель жизненного цикла —
-определение определенных действий, которые сопровождают изменения состояний объектов
-типичная схема последовательности работ на этапах разработки программного продукта
-отражение динамики изменений состояния каждого класса объектов

029. Понятность — это
-атрибут функциональности, указывающий на возможность предотвращать несанкционированный доступ

-атрибут надежности, который указывает на способность программы к перезапуску для повторного выполнения
-атрибут удобства, определяющий усилия, необходимые для распознавания логических концепций и условий их применения

030. Артефакт — это
-любой продукт деятельности специалистов по разработке программного обеспечения
-результат ошибок разработчика во входных или проектных спецификациях
-графическое представление элементов моделирования системы

Тематика докладов

1. Эволюция сложных программных систем
2. Методы документирования архитектуры
3. Управление знаниями в процессе разработки программных систем
4. CASE технологии разработки программных систем
5. Модели программных систем
6. Построение процесса разработки программных систем
7. Бизнес аспекты разработки программных систем
8. Модели ROI для оценки эффективности компаний-разработчиков программного обеспечения
9. Человеческий фактор при разработке ПО
10. Модели и методы оценки личностных характеристик исполнителей и команды в целом.
11. Оценка затрат программных проектов методом функциональных точек.
12. Регрессионная модель оценки затрат программных проектов COCOMO II.
13. Оценка программных проектов в модели SLIM.
14. Методы выбора организационной формы реализации программного проекта.
15. Количественные методики оценки рисков программных проектов.
16. Метрические показатели в оценке программных проектов.
17. Модели структурного анализа программных проектов.
18. Модели объектно-ориентированного анализа программных проектов.
19. Метод определения точек тестирования, основанный на анализе цикломатической сложности Мак-Кейба.
20. Сравнительный анализ инструментов моделирования и трассировки программных требований.
21. Сравнительный анализ инструментов верификации программных проектов.
22. Сравнительный анализ инструментов оптимизации программных проектов.
23. Сравнительный анализ инструментов тестирования программного обеспечения (генераторы тестов, схемы выполнения тестов, оценка тестов, управление тестами).
24. Сравнительный анализ инструментов сопровождения программного обеспечения.
25. Системы моделирования процессов разработки программного обеспечения.
26. Среды разработки программного обеспечения, ориентированные на процессы.
27. Сравнительный анализ инструментов обеспечения качества программного обеспечения.
28. Сравнительный анализ инструментов управления конфигурацией программного обеспечения.
29. Инструменты планирования и отслеживания программных проектов.
30. Инструменты, реализующие поддержку инфраструктуры разработки.

6. Учебно-методическое и информационное обеспечение дисциплины

6.1 Список источников и литературы

Литература

Основная

1. Черткова, Е. А. Программная инженерия. Визуальное моделирование программных систем : учебник для вузов / Е. А. Черткова. — 3-е изд., перераб. и доп. — Москва :

Издательство Юрайт, 2026. — 146 с. — (Высшее образование). — ISBN 978-5-534-18197-5. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/584957
2. Лаврищева, Е. М. Программная инженерия. Парадигмы, технологии и CASE-средства : учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. — Москва : Издательство Юрайт, 2026. — 280 с. — (Высшее образование). — ISBN 978-5-534-01056-5. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/584533
3. Управление программными проектами : учебник для вузов / под редакцией Р. Ф. Маликова. — Москва : Издательство Юрайт, 2026. — 167 с. — (Высшее образование). — ISBN 978-5-534-14329-4. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/588424

Дополнительная

1. Загорулько, Ю. А. Искусственный интеллект. Инженерия знаний : учебное пособие для вузов / Ю. А. Загорулько, Г. Б. Загорулько. — Москва : Издательство Юрайт, 2024. — 93 с. — (Высшее образование). — ISBN 978-5-534-07198-6. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/540987
2. Богатырев, В. А. Информационные системы и технологии. Теория надежности : учебное пособие для вузов / В. А. Богатырев. — 2-е изд. — Москва : Издательство Юрайт, 2024. — 366 с. — (Высшее образование). — ISBN 978-5-534-15951-6. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/510320
3. Трояновский, В. М. Программная инженерия информационно-управляющих систем в свете прикладной теории случайных процессов : учебное пособие / В.М. Трояновский. — Москва : ИНФРА-М, 2024. — 325 с. + Доп. материалы [Электронный ресурс]. — (Высшее образование: Магистратура). — DOI 10.12737/textbook_5ad88bf5c35cd8.81685342. - ISBN 978-5-8199-0824-2. - Текст : электронный. - URL: https://znanium.com/catalog/product/2059558

6.2 Перечень ресурсов информационно-телекоммуникационной сети «Интернет».

Национальная электронная библиотека (НЭБ) www.rusneb.ru
ELibrary.ru Научная электронная библиотека www.elibrary.ru
Электронная библиотека Grebennikon.ru www.grebennikon.ru
Cambridge University Press

6.3 Профессиональные базы данных и информационно-справочные системы

Доступ к профессиональным базам данных: https://liber.rsuh.ru/ru/bases
Информационные справочные системы:
1. Консультант Плюс
2. Гарант

7. Материально-техническое обеспечение дисциплины

№п/п	Наименование специальных помещений и помещений для самостоятельной работы	Оснащенность специальных помещений и помещений для самостоятельной работы	Перечень лицензионного программного обеспечения. Реквизиты подтверждающего документа		
			Наименование ПО	Лицензия/сертификат/заказ	Дата лицензии
1.	Лаборатория информатики – ауд. № 203	1 компьютер преподавателя, 12 компьютеров обучающихся, маркерная	Windows 7 Microsoft office 2010 Pro Microsoft	68526624 49420326 63202190 свободный доступ	без даты 08.12.2011 без даты свободный

		доска, проектор	Visual Professional 2019 Mozilla Firefox 52.8.1 ESR Matlab Mathcad Education - University edition Kaspersky Endpoint Security	647526 2996385 17E0181226094912873979	доступ без даты 14.06.2019 26.12.2018

8. Обеспечение образовательного процесса для лиц с ограниченными возможностями здоровья и инвалидов

В ходе реализации дисциплины используются следующие дополнительные методы обучения, текущего контроля успеваемости и промежуточной аттестации обучающихся в зависимости от их индивидуальных особенностей:

- для слепых и слабовидящих: лекции оформляются в виде электронного документа, доступного с помощью компьютера со специализированным программным обеспечением; письменные задания выполняются на компьютере со специализированным программным обеспечением или могут быть заменены устным ответом; обеспечивается индивидуальное равномерное освещение не менее 300 люкс; для выполнения задания при необходимости предоставляется увеличивающее устройство; возможно также использование собственных увеличивающих устройств; письменные задания оформляются увеличенным шрифтом; экзамен и зачёт проводятся в устной форме или выполняются в письменной форме на компьютере.
- для глухих и слабослышащих: лекции оформляются в виде электронного документа, либо предоставляется звукоусиливающая аппаратура индивидуального пользования; письменные задания выполняются на компьютере в письменной форме; экзамен и зачёт проводятся в письменной форме на компьютере; возможно проведение в форме тестирования.
- для лиц с нарушениями опорно-двигательного аппарата: лекции оформляются в виде электронного документа, доступного с помощью компьютера со специализированным программным обеспечением; письменные задания выполняются на компьютере со специализированным программным обеспечением; экзамен и зачёт проводятся в устной форме или выполняются в письменной форме на компьютере.

При необходимости предусматривается увеличение времени для подготовки ответа.

Процедура проведения промежуточной аттестации для обучающихся устанавливается с учётом их индивидуальных психофизических особенностей. Промежуточная аттестация может проводиться в несколько этапов.

При проведении процедуры оценивания результатов обучения предусматривается использование технических средств, необходимых в связи с индивидуальными особенностями обучающихся. Эти средства могут быть предоставлены университетом, или могут использоваться собственные технические средства.

Проведение процедуры оценивания результатов обучения допускается с использованием дистанционных образовательных технологий.

Обеспечивается доступ к информационным и библиографическим ресурсам в сети Интернет для каждого обучающегося в формах, адаптированных к ограничениям их здоровья и восприятия информации:

- для слепых и слабовидящих: в печатной форме увеличенным шрифтом, в форме электронного документа, в форме аудиофайла.
- для глухих и слабослышащих: в печатной форме, в форме электронного документа.

• для обучающихся с нарушениями опорно-двигательного аппарата: в печатной форме, в форме электронного документа, в форме аудиофайла.

Учебные аудитории для всех видов контактной и самостоятельной работы, научная библиотека и иные помещения для обучения оснащены специальным оборудованием и учебными местами с техническими средствами обучения:

• для слепых и слабовидящих: устройством для сканирования и чтения с камерой SARA CE; дисплеем Брайля PAC Mate 20; принтером Брайля EmBraille ViewPlus;

• для глухих и слабослышащих: автоматизированным рабочим местом для людей с нарушением слуха и слабослышащих; акустический усилитель и колонки;

• для обучающихся с нарушениями опорно-двигательного аппарата: передвижными, регулируемыми эргономическими партами СИ-1; компьютерной техникой со специальным программным обеспечением.

9. Методические материалы

9.1 Планы практических занятий

В плане практических занятий выполняются следующие работы:

ПРАКТИЧЕСКАЯ РАБОТА №1

ОЗНАКОМЛЕНИЕ С RATIONAL ROSE 2000. ФОРМИРОВАНИЕ ДИАГРАММ USE CASE

Порядок выполнения работы

Для создания новой пустой модели необходимо запустить Rational Rose 2000 и нажать кнопку Cancel в окне, показанном на рис. 1.

Рис. 1. Окно выбора разновидности новой модели

В дальнейших работах следует не создавать новые файлы моделей, а работать в файле,

созданном в первой работе, добавляя в него новые модели.

Создание диаграммы Use Case (вариантов использования) производится через пункты главного меню BrowseàUse Case Diagram… Создание новых элементов Use Case возможно при помощи команд меню ToolsàCreate или при помощи панели инструментов (рис. 2).

Назначение диаграмм Use Case:

1. Определение поведения системы с точки зрения пользователя.
2. Первичное моделирования динамики системы.
3. Выяснение требований к разрабатываемой системе.
4. Фиксация этих требований в унифицированной форме.

Состав диаграмм Use Case:

1) Элементы Use Case.
2) Актеры.
3) Отношения:
   - Зависимости;
   - Обобщения;
   - Ассоциации.
4) Примечания.
5) Ограничения.
6) Пакеты для группировки элементов модели.

Инструмент выбора (Selection Tool)
Текст (Text Box)
Комментарий (примечание, Note)
Якорь для замечания (Note Anchor)
Контейнер (пакет, Package)
Сценарии поведения (Use Case)
Актер (Actor)
Однонаправленная связь (ассоциация, Unidirectional Association)
Зависимость (Dependency)
Наследование-обобщение (Generalization)

Рис. 2. Панель инструментов для диаграммы Use Case

Актер – роль объекта вне системы, который прямо взаимодействует с ее частью (конкретным элементом Use Case). Пользователь – это физический объект, использующий систему. Он может играть несколько ролей и может моделироваться несколькими актерами. Обратное верно: актером могут быть разные пользователи. В ИС, представляющих собой аппаратно-программные комплексы, актерами могут быть не только пользователи, но и аппаратные компоненты, являющиеся внешними для программной системы.

Элемент Use Case – описание последовательности действий (нескольких последовательностей), которые выполняются системой и производят для отдельного актера видимый результат.

Один актер может использовать несколько элементов Use Case. Один элемент Use Case может использоваться несколькими актерами. Каждый элемент Use Case описывает определенный путь использования системы. Набор всех элементов Use Case определяет полные функциональные возможности системы.

Между актером и элементом Use Case возможно только отношение ассоциации. Ассоциация может быть отмечена именем, ролями, мощностью. Прочие разновидности отношений в рамках настоящей практической работы рассматриваться не будут.

В качестве примера рассмотрим следующую задачу.

Пусть требуется разработать программу, демонстрирующую обработку одномерного массива. Обрабатывающая процедура в зависимости от настроек должна делить или умножать все элементы массива на заданное число. Кроме того, должны быть предусмотрены процедуры

получения массива путем ручного ввода пользователем, автоматической генерации или загрузки из дискового файла, а также сохранения массива в файле.

Пользователю должны быть предоставлены возможности:

1) Генерации массива:
   - автоматического создания массива;
   - ручного ввода массива;
   - сохранения массива;
   - загрузки массива.
2) Обработки массива.

В результате анализа сформулированного задания получаем (с использованием инструментов Use Case, Actor и Unidirectional Association) диаграмму, показанную на рис. 3.

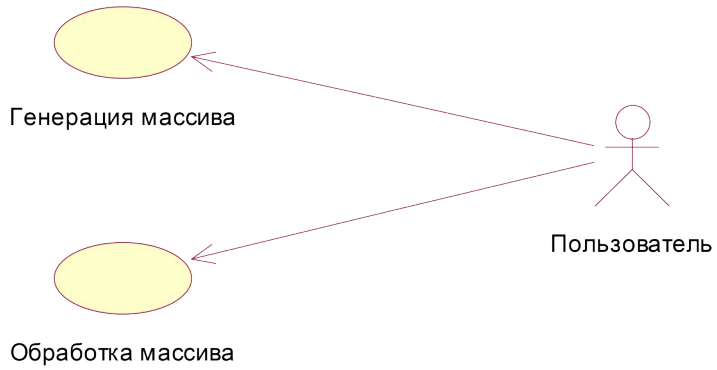


Рис. 3. Диаграмма Use Case программного средства демонстрации обработки одномерного массива

По каждому элементу Use Case следует ввести первоначальную документацию, т.е. отметить основные сценарии, входящие в элемент. Это можно сделать в диалоговом окне спецификации элемента Use Case, вызываемом через пункт “Open Specification…” контекстного меню элемента Use Case (рис. 4).

Указанные сценарии разрабатываются, т.е. излагаются в словесном виде без технических подробностей (рис. 6). Эти действия нужно выполнить для каждого элемента Use Case.

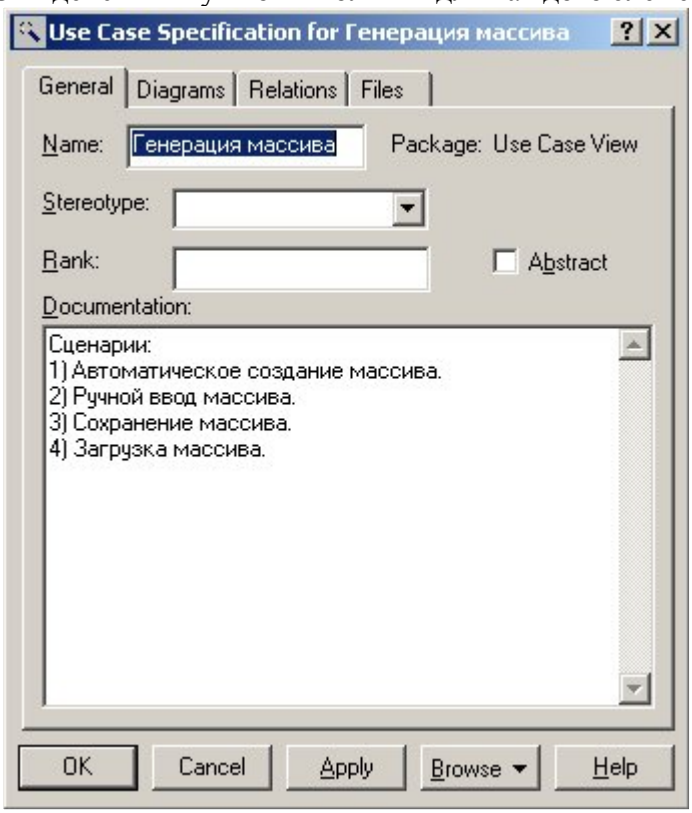


Рис. 4. Спецификация элемента Use Case Генерация массива

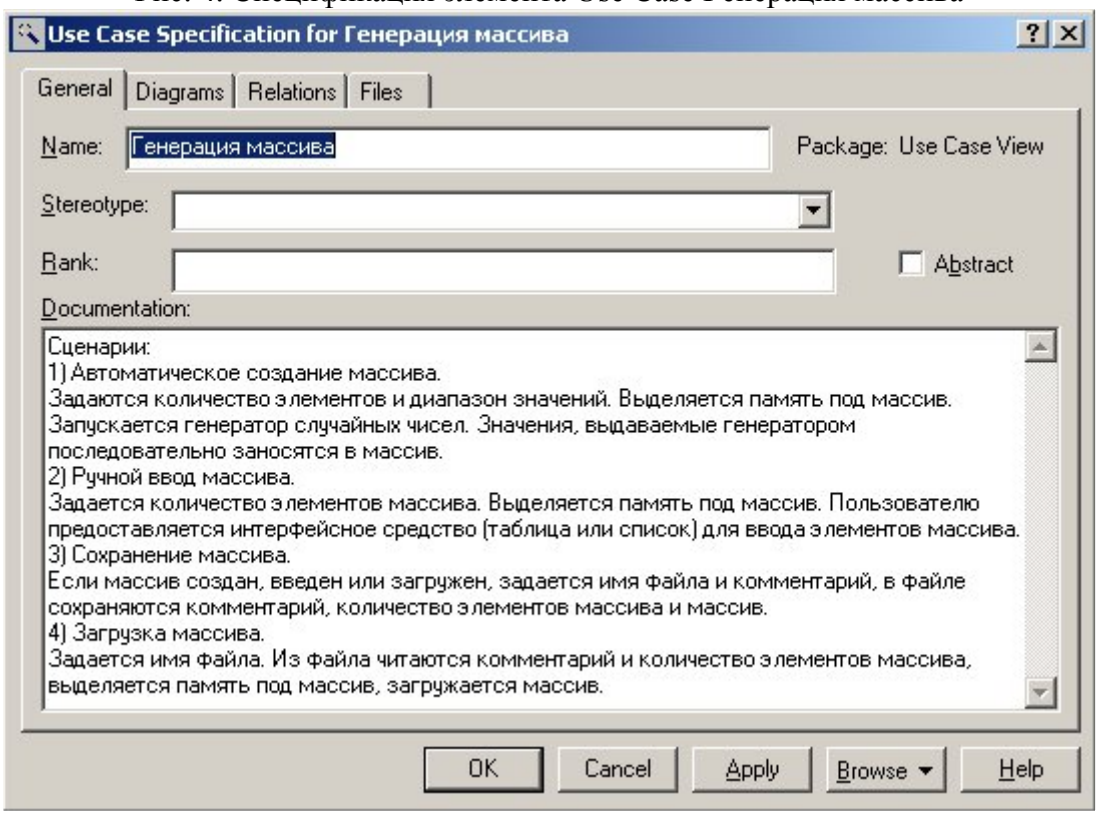


Рис. 5. Спецификация элемента Use Case Генерация массива со сценариями

Варианты заданий

Вариант 1. Программное средство демонстрации анализа корреляционных зависимостей

Разработать программу, демонстрирующую анализ корреляционных зависимостей между двумя рядами данных.

Пользователю должны быть предоставлены возможности:

1) Генерации ряда:
   - автоматического создания ряда (задаются количество отсчетов, частота дискретизации, количество гармоник ряда, амплитуды и частоты для каждой гармоники, амплитуда случайной составляющей; выделяется память под массив; массив заполняется генерируемыми значениями; созданный сигнал отображается на графике и в таблице);
   - сохранения ряда (если ряд присутствует, задается имя файла, в файл записываются параметры ряда и ряд);
   - загрузки ряда (задается имя файла, из файла загружаются параметры ряда и ряд).
2) Обработки рядов (рассчитывается корреляционная функция в диапазоне от –T/2 до T/2, где T – длина минимального из двух рядов, корреляционная функция выводится в таблицу и на график).

Вариант 2. Программное средство демонстрации гармонического анализа

Разработать программу, демонстрирующую гармонический анализ ряда.

Пользователю должны быть предоставлены возможности:

1) Генерации ряда:
   - автоматического создания ряда (задаются количество отсчетов, частота дискретизации, количество гармоник ряда, амплитуды и частоты для каждой гармоники, амплитуда случайной составляющей; выделяется память под массив; массив заполняется генерируемыми значениями; созданный сигнал отображается на графике и в таблице);
   - сохранения ряда (если ряд присутствует, задается имя файла, в файл записываются параметры ряда и ряд);
   - загрузки ряда (задается имя файла, из файла загружаются параметры ряда и ряд).

2) Обработки рядов (рассчитывается ряд Фурье и спектр мощности, спектр мощности выводится на график и в таблицу).

Вариант 3. Программное средство демонстрации расчета сезонной составляющей ряда

Разработать программу, демонстрирующую расчет и использование сезонной составляющей ряда.

Пользователю должны быть предоставлены возможности:

1) Генерации ряда:
   - автоматического создания ряда (задаются количество отсчетов, значения сезонной составляющей, постоянная составляющая, амплитуда случайной составляющей; выделяется память под массив; массив заполняется генерируемыми значениями; созданный сигнал отображается на графике и в таблице);
   - сохранения ряда (если ряд присутствует, задается имя файла, в файл записываются параметры ряда и ряд);
   - загрузки ряда (задается имя файла, из файла загружаются параметры ряда и ряд).
2) Расчета модели ряда (рассчитываются сезонные составляющие).
3) Построения модельного ряда (рассчитываются отсчеты ряда по модельным значениям сезонной составляющей, полученный ряд отображается в виде таблицы и графика).

Вариант 4. Программное средство демонстрации расчета авторегрессионной модели ряда

Разработать программу, демонстрирующую расчет и использование авторегрессионной модели ряда.

Пользователю должны быть предоставлены возможности:

1) Генерации ряда:
   - автоматического создания ряда (задаются количество отсчетов, частота дискретизации, количество гармоник ряда, амплитуды и частоты для каждой гармоники, амплитуда случайной составляющей; выделяется память под массив; массив заполняется генерируемыми значениями; созданный сигнал отображается на графике и в таблице);
   - сохранения ряда (если ряд присутствует, задается имя файла, в файл записываются параметры ряда и ряд);
   - загрузки ряда (задается имя файла, из файла загружаются параметры ряда и ряд).
2) Расчета модели ряда (рассчитываются коэффициенты авторегрессии).
3) Построения модельного ряда (рассчитываются отсчеты ряда в соответствии с авторегрессионной моделью, полученный ряд отображается в виде таблицы и графика).

Вариант 5. Программное средство демонстрации расчета характеристик ряда и построения распределений

Разработать программу, демонстрирующую расчет моментных характеристик (среднего значения и среднеквадратичного отклонения) ряда и построения распределения.

Пользователю должны быть предоставлены возможности:

1) Генерации ряда:
   - автоматического создания ряда (задаются количество отсчетов, амплитуда случайной составляющей; выделяется память под массив; массив заполняется генерируемыми значениями; созданный ряд отображается на графике и в таблице);
   - сохранения ряда (если ряд присутствует, задается имя файла, в файл записываются параметры ряда и ряд);
   - загрузки ряда (задается имя файла, из файла загружаются параметры ряда и ряд).
2) Анализа ряда (рассчитываются среднее значение и среднеквадратичное отклонение, максимальное и минимальное значения ряда; в виде таблицы и графика выводятся значения распределения значений ряда по десяти поддиапазонам всего диапазона значений).

Вариант 6. Программное средство поисковой обработки текстов

Разработать программу, выполняющую поиск в тексте заданных строк.

Пользователю должны быть предоставлены возможности:

1) Получения текста:

- ручного набора;
- сохранения текста (если текст присутствует, задается имя файла, в файл записываются текст);
- загрузки текста (задается имя файла, из файла загружается текст).

2) Анализа текста (задается набор искомых строк, производится обработка текста с запоминанием позиций найденных строк; организуется просмотр найденных фрагментов).

Вариант 7. Программное средство корректирующей обработки текстов

Разработать программу, выполняющую замены фрагментов текста в соответствии с установками пользователя.

Пользователю должны быть предоставлены возможности:

1) Получения текста:
   - ручного набора;
   - сохранения текста (если текст присутствует, задается имя файла, в файл записываются текст);
   - загрузки текста (задается имя файла, из файла загружается текст).
2) Обработки текста (указывается тип обработки – заменить заданное слово, заменить все строчные буквы на прописные, заменить все прописные буквы на строчные; для первого типа обработки задаются заменяемая и заменяющая строка; производится обработка текста).

Вариант 8. Программное средство статистической обработки текстов

Разработать программу, выполняющую анализ текста и подсчет статистик.

Пользователю должны быть предоставлены возможности:

1) Получения текста:
   - ручного набора;
   - сохранения текста (если текст присутствует, задается имя файла, в файл записываются текст);
   - загрузки текста (задается имя файла, из файла загружается текст).
2) Обработки текста (указывается тип обработки – подсчет количества слов, расчет количеств и частот встречаемости букв; производится обработка в соответствии с заданным типом и отображение результатов; для рассчитанного распределения по буквам строится гистограмма).

Вариант 9. Программное средство сортировки массивов

Разработать программу, выполняющую сортировку массива в соответствии с установками пользователя.

Пользователю должны быть предоставлены возможности:

1) Генерации массива:
   - автоматического создания массива (задаются количество элементов массива, диапазон значений случайных чисел; выделяется память под массив; массив заполняется генерируемыми значениями; созданный массив отображается в таблице);
   - ручного ввода массива (задается количество элементов массива; осуществляется ввод элементов; введенный массив отображается в таблице);
   - сохранения массива (если массив присутствует, задается имя файла, в файл записывается массив);
   - загрузки массива (задается имя файла, из файла загружается массив).
2) Сортировки массива (задается тип сортировки – по возрастанию или по убыванию; выполняется сортировка).

Вариант 10. Программное средство сортировки двумерных массивов

Разработать программу, выполняющую сортировку двумерного массива в соответствии с установками пользователя.

Пользователю должны быть предоставлены возможности:

1) Генерации массива:
   - автоматического создания массива (задаются количество строк и столбцов массива,

диапазон значений случайных чисел; выделяется память под массив; массив заполняется генерируемыми значениями; созданный массив отображается в таблице);
- ручного ввода массива (задается количество строк и столбцов массива; осуществляется ввод элементов; введенный массив отображается в таблице);
- сохранения массива (если массив присутствует, задается имя файла, в файл записываются количество строк и столбцов массива и массив);
- загрузки массива (задается имя файла, из файла загружаются количество строк и столбцов массива и массив).

2) Сортировки массива (задается тип сортировки – по возрастанию или по убыванию, и параметр сортировки – минимальное значение в строке, максимальное значение в строке, сумма элементов строки, первый элемент строки, последний элемент строки; выполняется сортировка строк массива).

Вариант 11. Программное средство обработки массивов

Разработать программу, выполняющую обработку массива в соответствии с установками пользователя.

Пользователю должны быть предоставлены возможности:

1) Генерации массива:
   - автоматического создания массива (задаются количество элементов массива, диапазон значений случайных чисел; выделяется память под массив; массив заполняется генерируемыми значениями; созданный массив отображается в таблице);
   - ручного ввода массива (задается количество элементов массива; осуществляется ввод элементов; введенный массив отображается в таблице);
   - сохранения массива (если массив присутствует, задается имя файла, в файл записывается массив);
   - загрузки массива (задается имя файла, из файла загружается массив).
2) Обработки массива (задается тип обработки – заменить все отрицательные элементы нулевыми, заменить все элементы, большие среднего значения, средним значением, сменить знаки у всех отрицательных чисел; выполняется обработка).

Вариант 12. Программное средство обработки двумерных массивов

Разработать программу, выполняющую обработку двумерного массива в соответствии с установками пользователя.

Пользователю должны быть предоставлены возможности:

1) Генерации массива:
   - автоматического создания массива (задаются количество строк и столбцов массива, диапазон значений случайных чисел; выделяется память под массив; массив заполняется генерируемыми значениями; созданный массив отображается в таблице);
   - ручного ввода массива (задается количество строк и столбцов массива; осуществляется ввод элементов; введенный массив отображается в таблице);
   - сохранения массива (если массив присутствует, задается имя файла, в файл записываются количество строк и столбцов массива и массив);
   - загрузки массива (задается имя файла, из файла загружаются количество строк и столбцов массива и массив).
2) Обработки массива (задается тип обработки – заменить в каждой строке первый элемент средним значением строки, заменить в каждом столбце все элементы средним значением столбца; выполняется обработка массива).

ПРАКТИЧЕСКАЯ РАБОТА №2

РАЗРАБОТКА ДИАГРАММЫ ПОСЛЕДОВАТЕЛЬНОСТИ (SEQUENCE)

Порядок выполнения работы

Диаграммы последовательности предназначены для отображения порядка обмена

сообщениями между объектами системы.

В этой и последующих работах используется файл модели, созданный в первой работе. Не нужно создавать новый файл.

Создание диаграммы последовательности выполняется тремя способами.

Способ 1. В окне Browser (рис. 6) в контекстном меню папки Logical View следует выбрать NewàSequence Diagram, а затем ввести имя новой диаграммы.

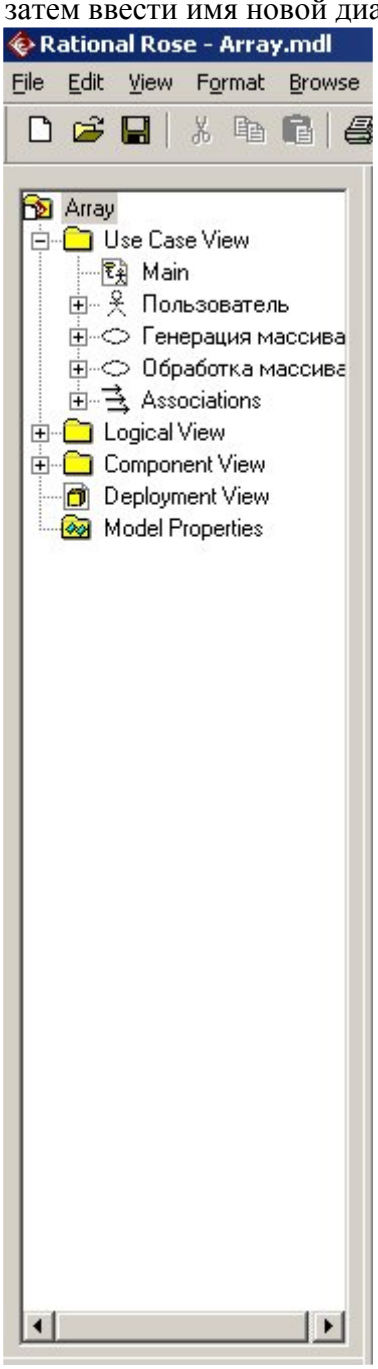


Рис. 6. Вид окна Browser после создания диаграммы Use Case

Способ 2. В главном меню программы следует выбрать BrowseàInteraction Diagram…, после чего выбрать пункт <New> (или выбрать имя диаграммы при необходимости редактирования созданной ранее), а затем в диалоговом окне (рис. 7) задать имя и тип диаграммы. Это диалоговое окно является общим для обоих типов диаграмм взаимодействий – и для диаграмм последовательности, и для диаграмм кооперации, поэтому в данном случае необходимо выбрать тип Sequence для разработки диаграммы последовательности.

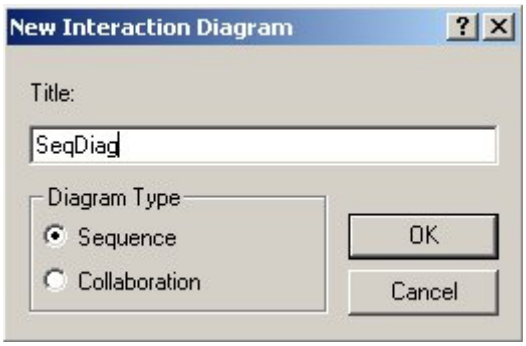


Рис. 7. Диалоговое окно ввода имени и типа диаграммы автоматов

Способ 3. Этот способ является наиболее естественным и интуитивно понятным при последовательном моделировании работы системы. После создания Use Case диаграмм и словесного описания сценариев, реализуемых Use Case элементами, каждая из диаграмм формализуется при помощи одной или набора диаграмм последовательностей. В окне Browser в контекстном меню соответствующего элемента Use Case из папки Use Case View следует выбрать NewàSequence Diagram, а затем ввести имя новой диаграммы.

Предлагается воспользоваться именно третьим способом.

Созданные диаграммы доступны из окна Browser. Панель инструментов диаграммы последовательности показана на рис. 8.

Инструмент выбора (Selection Tool)
Текст (Text Box)
Комментарий (примечание, Note)
Якорь для замечания (Note Anchor)
Объект (Object)
Сообщение (Object Message)
Сообщение самому себе (Message to Self)

Рис. 8. Панель инструментов для диаграммы последовательности

На диаграмме размещают объекты. Это делается при помощи инструмента Объект, а также, при наличии ранее созданных актеров в папке Use Case View или классов в папке Logical View, путем перетаскивания из окна Browser. Вид диаграммы после размещения на ней объектов показан на рис. 9. Здесь Пользователь выглядит как безымянный объект, обозначенный классом, что логически корректно. Объекты Массив и Таблица, напротив, не имеют имен классов, так как соответствующие классы еще не спроектированы.

Рис. 9. Начальный вид диаграммы последовательности

При помощи инструмента Сообщение создаются сообщения между линиями жизни объектов (рис. 10).

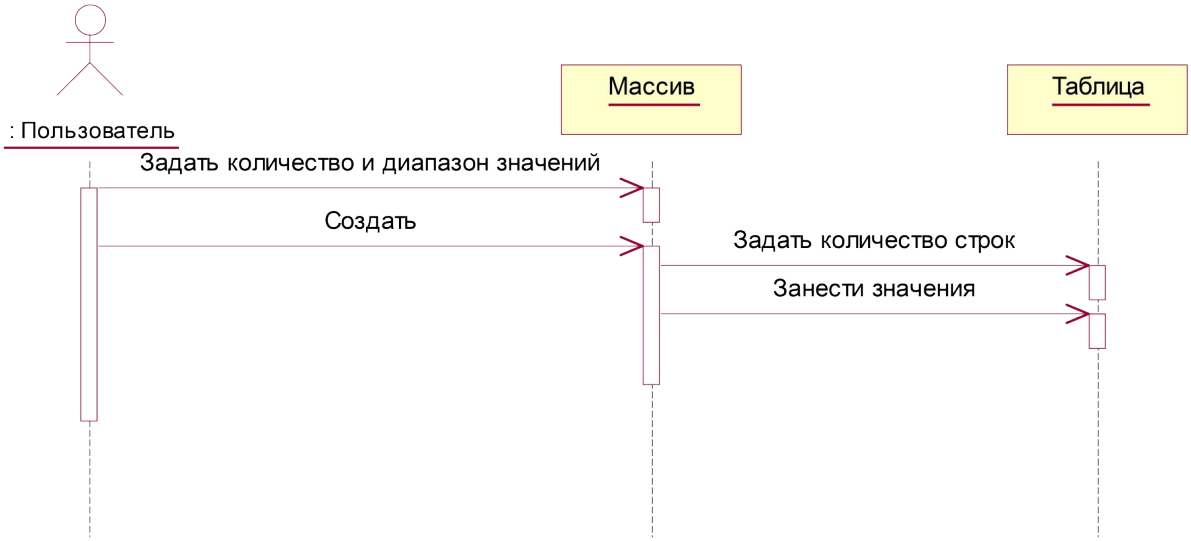


Рис. 10. Создание сообщений между линиями жизни объектов

Настройка сообщения (диалоговое окно Message Specification, рис. 11), кроме имени (вкладка General) включает настройку частоты и порядка обмена сообщениями (вкладка Detail).

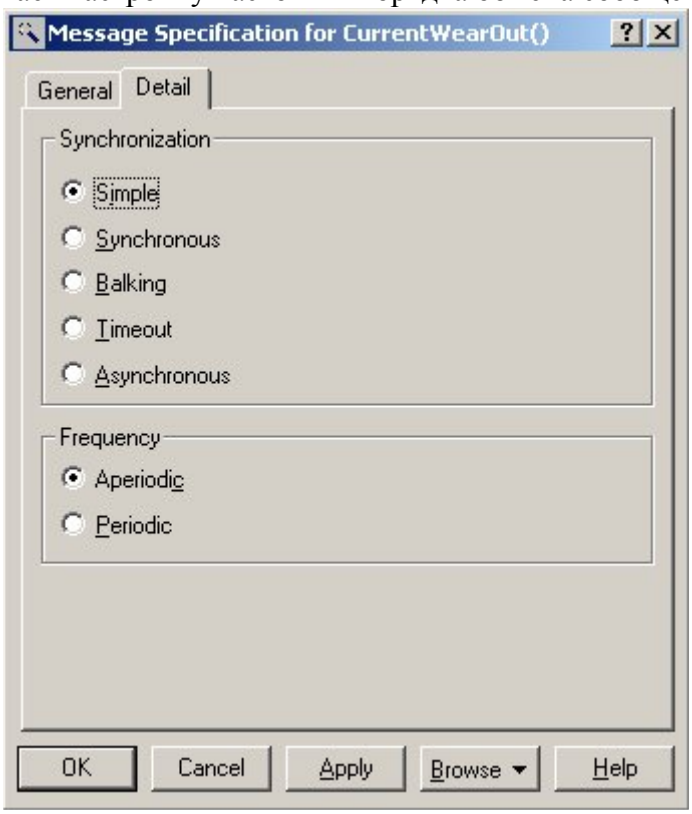


Рис. 11. Диалоговое окно спецификации сообщения

Варианты порядка обмена сообщениями:

- Simple – простая посылка сообщения;
- Synchronous – операция выполняется только в том случае, когда сервер может принять сообщение клиента;
- Balking – если сервер не готов немедленно принять сообщение, клиент отказывается от его посылки;
- Timeout – клиент отказывается от посылки сообщения, если сервер не готов его принять в течение некоторого времени;
- Asynchronous – клиент выдает сообщение и не ожидает ответа сервера.

Варианты настройки периодичности:

- Periodic – существует некоторая периодичность выдачи сообщений;
- Aperiodic – периодичность отсутствует.

Все эти настройки влияют только на вид стрелки сообщения.

На рис. 12 приведена также диаграмма обработки массива.

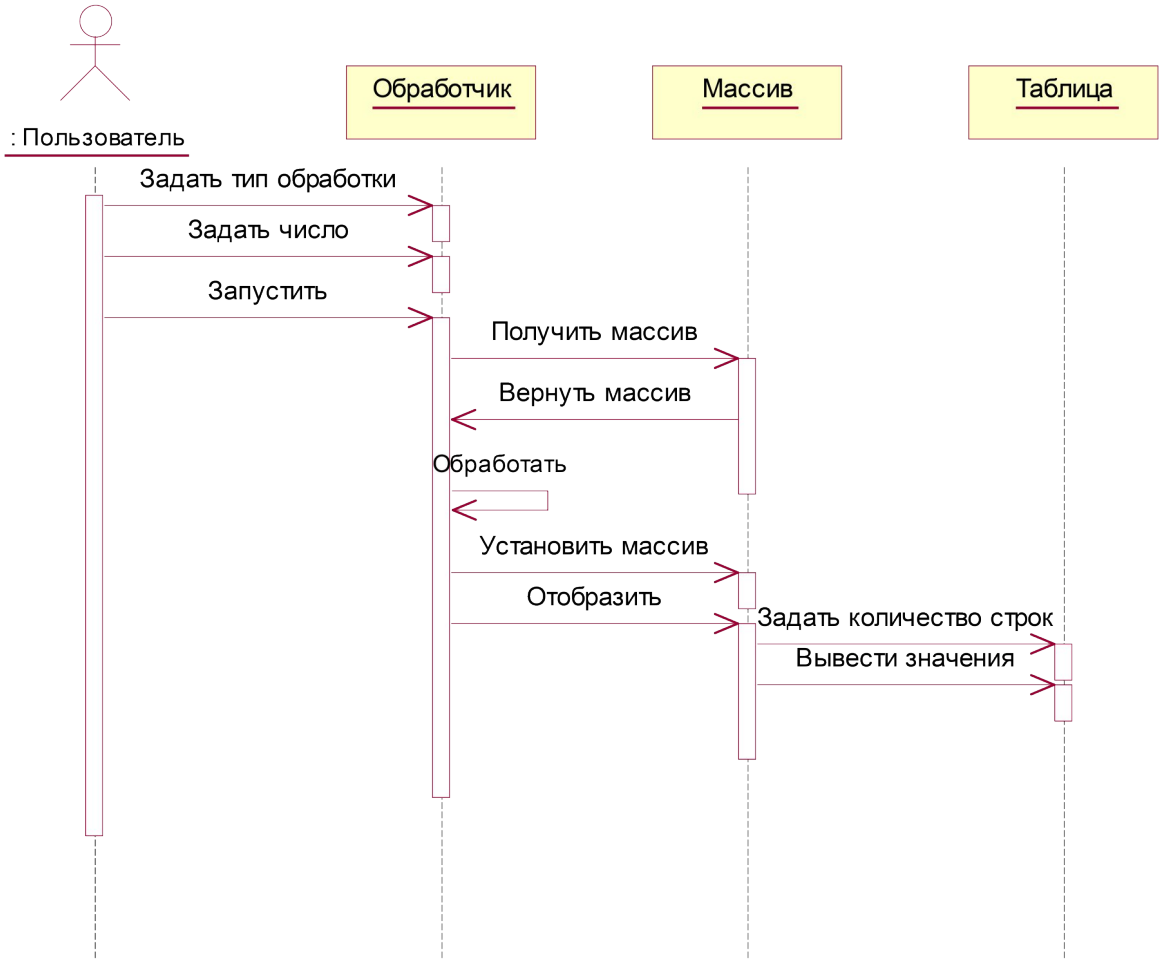


Рис. 12. Диаграмма последовательности обработки массива

В рамках настоящей работы следует изобразить в виде диаграмм последовательности все сценарии всех Use Case элементов согласно варианту задания.

ПРАКТИЧЕСКАЯ РАБОТА №3

РАЗРАБОТКА ДИАГРАММЫ КЛАССОВ

Порядок выполнения работы

Общий порядок функционирования объектов ИС определяется диаграммой последовательности, разработанной в ходе предыдущей практической работы. Следующий шаг – разработка структуры классов и сопоставление ее с объектами, обозначенными в диаграмме последовательности.

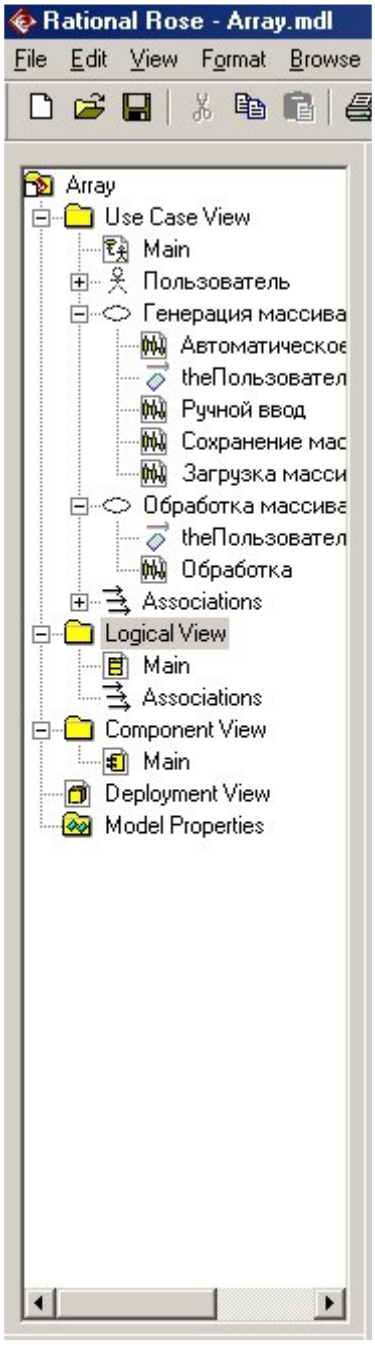
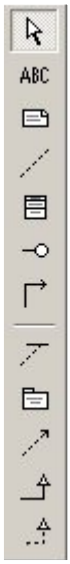


Рис. 13. Вид окна Browser перед созданием диаграммы классов

Несмотря на универсальность моделей Rational Rose, определимся с ориентацией на средство разработки Borland C++ Builder. В этом случае объект, названный в диаграмме последовательности «Таблица», реализуется при помощи стандартного визуального компонента класса TStringGrid. Таким образом, нужно реализовать классы, экземпляры которых были обозначены на этапе разработки диаграммы последовательности. Пусть в нашем случае экземпляры классов получили названия «Массив» и «Обработчик». Назовем соответствующие классы “Array” и “Processor”. Не следует давать классу русское имя, поскольку имя класса будет выгружаться в генерируемые тексты на языке программирования.

Для создания диаграммы последовательности следует выбрать в окне Browser (рис. 13) двойным щелчком Logical ViewàMain.



Инструмент выбора (Selection Tool)
Текст (Text Box)
Комментарий (примечание, Note)
Якорь для замечания (Note Anchor)
Класс (Class)
Интерфейс (Interface)
Однонаправленная ассоциация (Unidirectional Association)

Класс ассоциации (Association Class)
Контейнер (Package)
Зависимость реализаций (Dependency of instantiates)
Генерализация, обобщение, наследование (Generalization)
Реализация (Realize)

Рис. 14. Панель инструментов для диаграммы классов

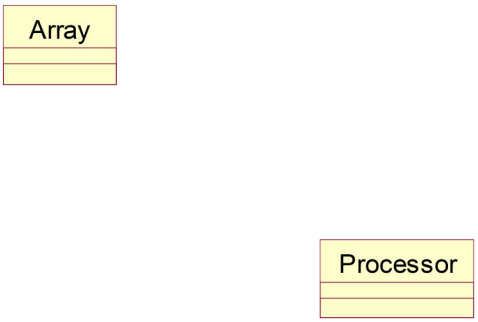


Рис. 15. Созданные классы

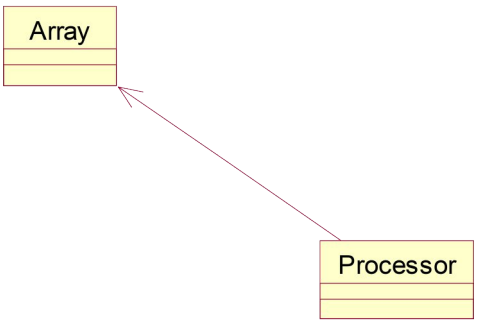


Рис. 16. Создание отношения ассоциации между классами

Новый класс создается при помощи панели инструментов окна Class Diagram (рис. 14). Для этого используется инструмент Class. Имя созданного класса вводится с клавиатуры. В результате получается заготовка нового класса, у которого остается задать свойства и операции (рис. 15).

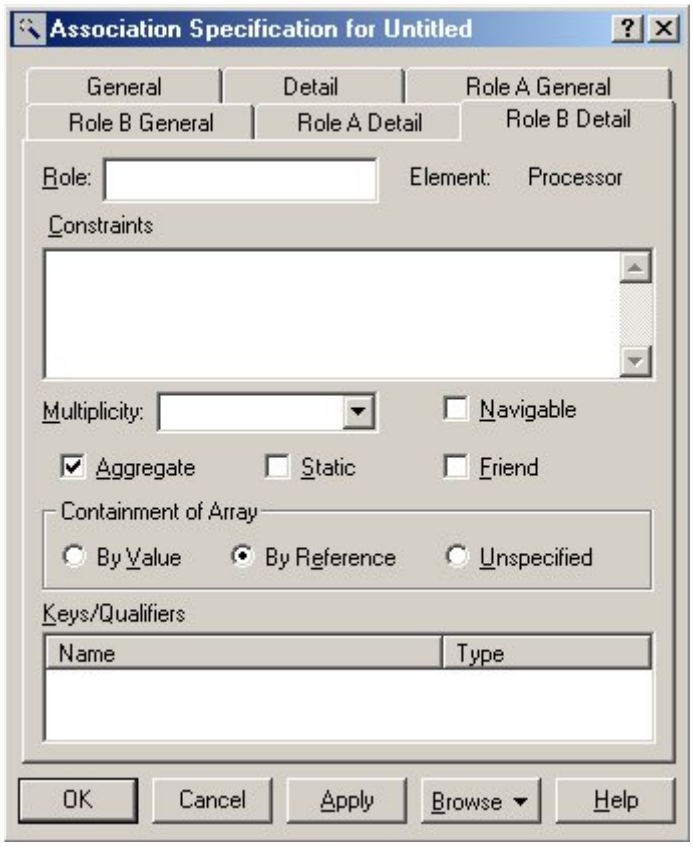


Рис. 17. Настройка агрегации

Однако перед заданием свойств и операций определим отношения между классами на диаграмме классов. В данном случае класс Processor является клиентом по отношению к классу Array, т.к. использует его операции. Кроме того, класс Processor должен хранить ссылку на класс Array, что соответствует отношению агрегации. Для настройки отношения используется инструмент Unidirectional Association (рис. 14). При этом стрелку направляют от клиента к серверу (рис. 16).

Настройка ассоциации выполняется либо через пункт Open Specification… ее контекстного меню, либо через элемент списка Relations диалогового окна спецификации класса. В данном случае необходимо только настроить один из классов, входящих в ассоциацию на агрегацию в себе другого класса по ссылке. Для этого перейдем на страницу Role B Detail (рис. 17). Ошибиться практически невозможно – в правом верхнем углу страницы указано имя класса, который настраивается как агрегирующий. Установим флажок Aggregate и переключатель By Reference. Полученное изображение отношения показано на рис. 18.

Свойства выявляются путем анализа диаграмм последовательности, а также диаграмм Use Case и сценариев.

У класса Array (Массив) должны быть следующие свойства:

- Ptr – указатель на массив;
- Num – количество элементов массива;
- FileName – имя файла;
- Grid – указатель на объект класса TStringGrid, используемый для отображения массива;
- Min и Max – нижняя и верхняя границы диапазона значений массива (используются при автоматической генерации элементов массива);
- Status – состояние массива (сгенерирован, загружен, сохранен, изменен).

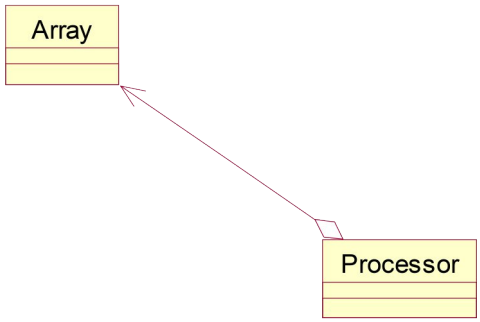


Рис. 18. Ассоциация, настроенная как агрегация

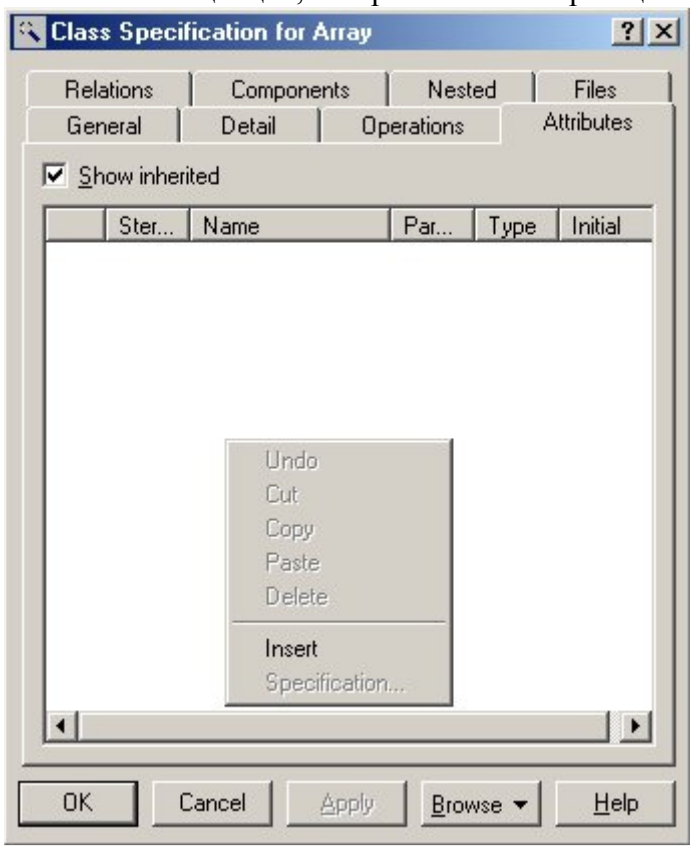


Рис. 19. Добавление свойств в класс

У класса Processor (Обработчик) должны быть следующие свойства:

- Value – значение, на которое следует умножать или делить обрабатываемый массив;
- ProcType – тип обработки.

Кроме того, должна быть ссылка на объект класса Array, но она задана в неявном виде отношением агрегации. Ее наличие будет очевидно на этапе генерации кода.

Для создания свойств класса в диалоговом окне спецификации класса (рис. 19) на вкладке Attributes следует выбрать пункт Insert в контекстном меню списка свойств. В появившейся строке введем имя свойства Ptr и при помощи двойного щелчка мышью войдем в диалоговое окно настройки свойства (рис. 20).

На вкладке General доступны для настройки тип значения свойства и видимость свойства – группа Export Control. Область видимости по умолчанию – приватная, и нет необходимости ее изменять. Тип значения также будет настроен позже – после настройки на язык программирования.

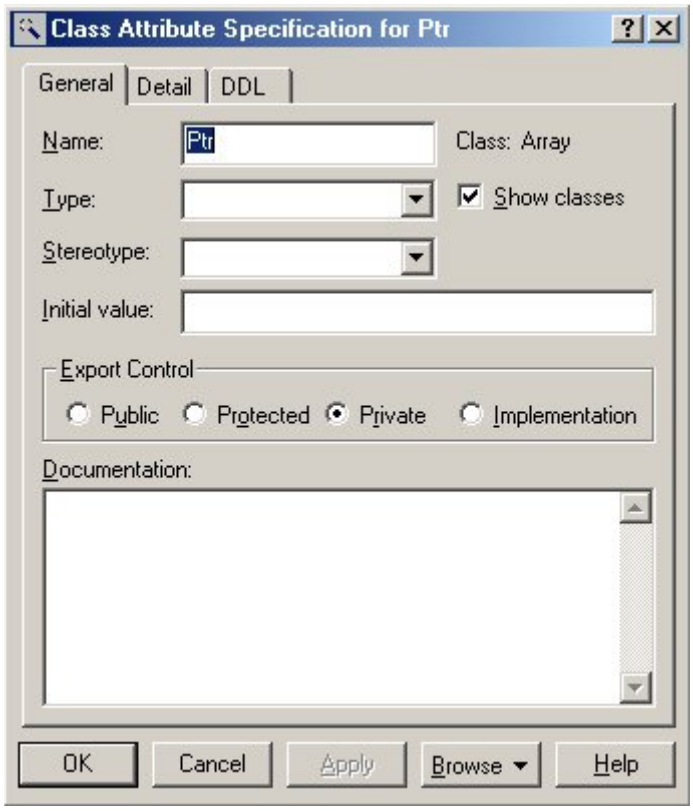


Рис. 20. Диалоговое окно спецификации свойства

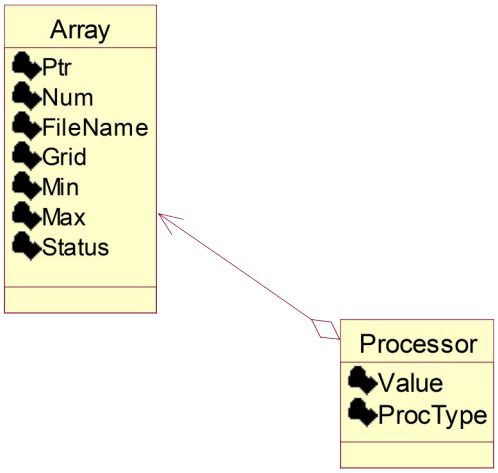


Рис. 21. Диаграмма классов с указанными свойствами

В результате полученная структура будет выглядеть, как показано на рис. 21.

Операции класса можно задать как непосредственно путем редактирования спецификации класса в диаграмме классов, так и проассоциировав класс с объектом диаграммы последовательности, а операции класса – с сообщениями диаграммы последовательности. Второй подход призван оптимизировать формирование набора операций класса и устранять случаи избыточностей и недоработок.

Это делается следующим образом. Необходимо открыть диаграмму последовательности обработки массива (рис. 12). Затем из папки Logical View окна Browser перетащить имя класса на соответствующий объект диаграммы последовательности. Например, класс Array перетащить на объект Массив, класс Processor – на объект Обработчик. В результате объекты на диаграмме последовательности будут изображены с указанием имен классов в соответствии с нотацией UML (рис. 22).

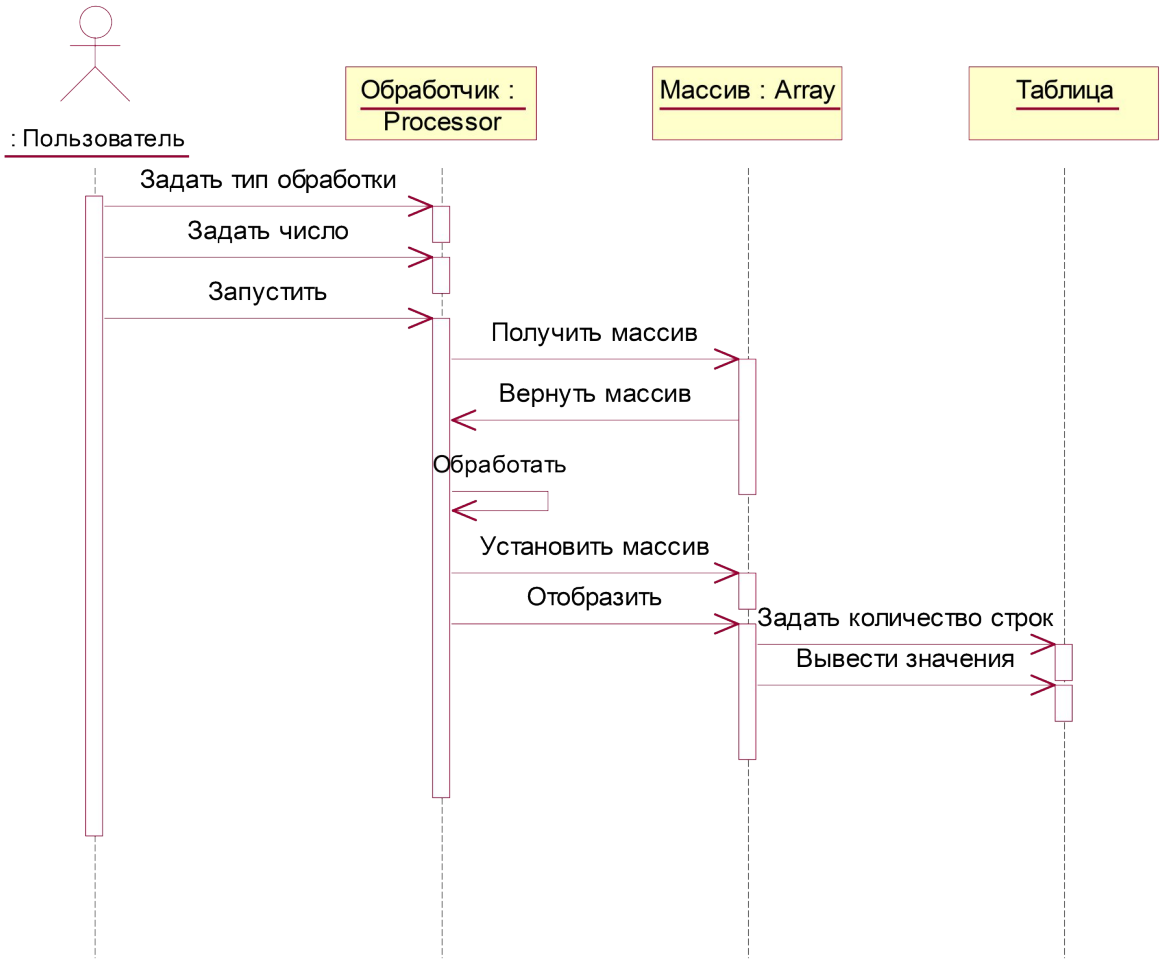


Рис. 22. Диаграмма последовательности с указанием классов

Теперь каждое сообщение, передаваемое объекту, для которого указан класс, может быть заменено именем операции этого класса, введенной ранее или создаваемой «на лету». Например, создадим операцию, вызываемую для задания типа обработки обработчика. Для этого необходимо вызвать контекстное меню сообщения «Задать тип обработки» (рис. 23) и выбрать из него пункт “<new operation>”, после чего отредактировать спецификацию операции (рис. 24). На данном этапе так же, как при редактировании свойств класса, достаточно ограничиться заданием имени операции. После этого сообщение на диаграмме последовательности приобретает вид рис. 25. Обратите внимание, как изменяется при этом описание классов в диаграмме классов.

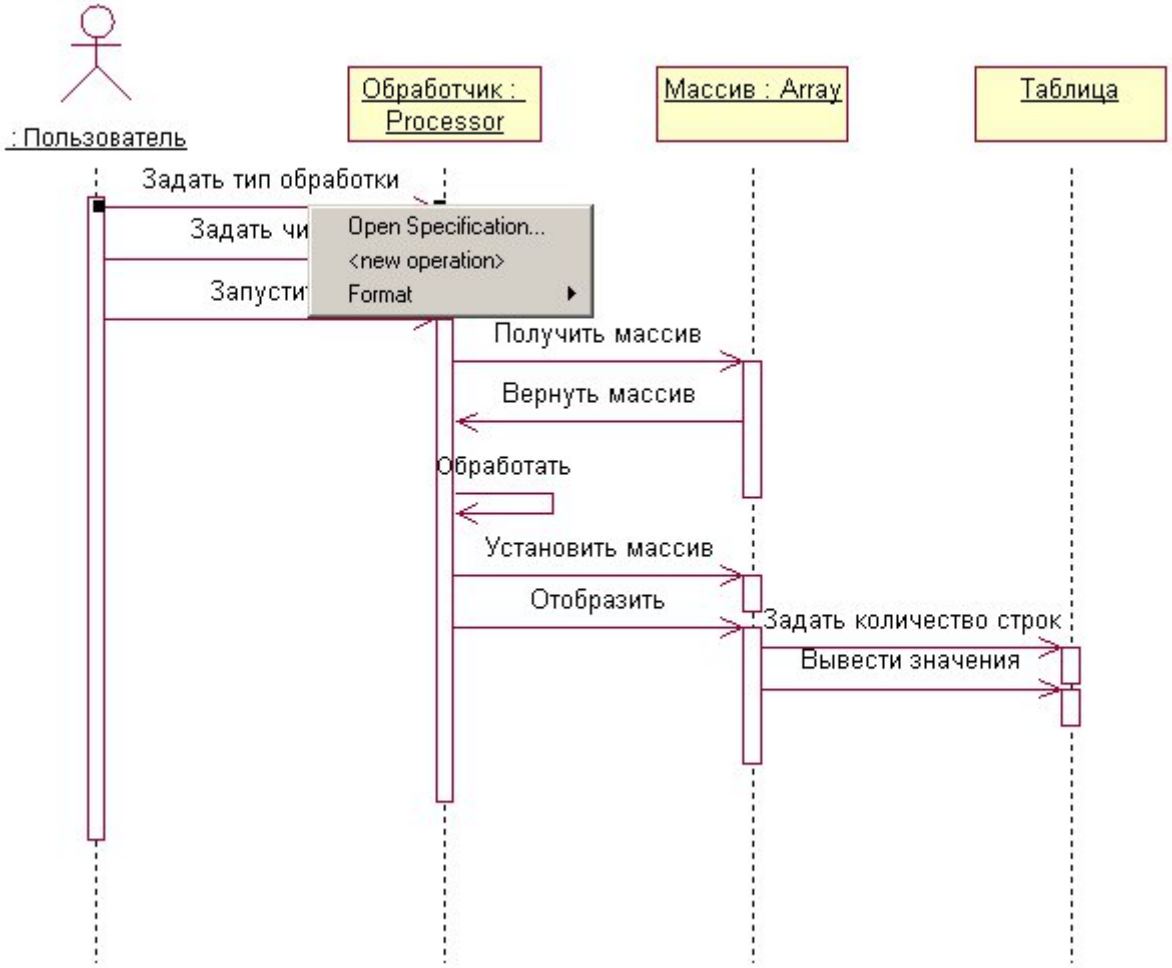


Рис. 23. Контекстное меню сообщения диаграммы последовательности

В рамках настоящей работы следует в соответствии с вариантом задания разработать диаграмму классов, сформировать наборы свойств и операций классов, связать объекты диаграмм последовательности с классами, а сообщения диаграмм последовательности – с операциями классов.

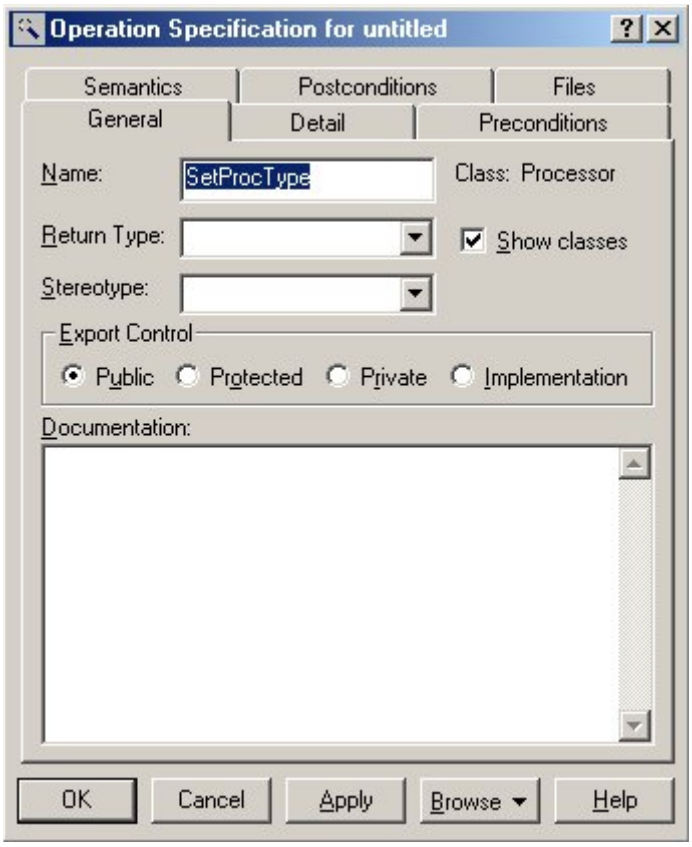


Рис. 24. Редактирование спецификации операции

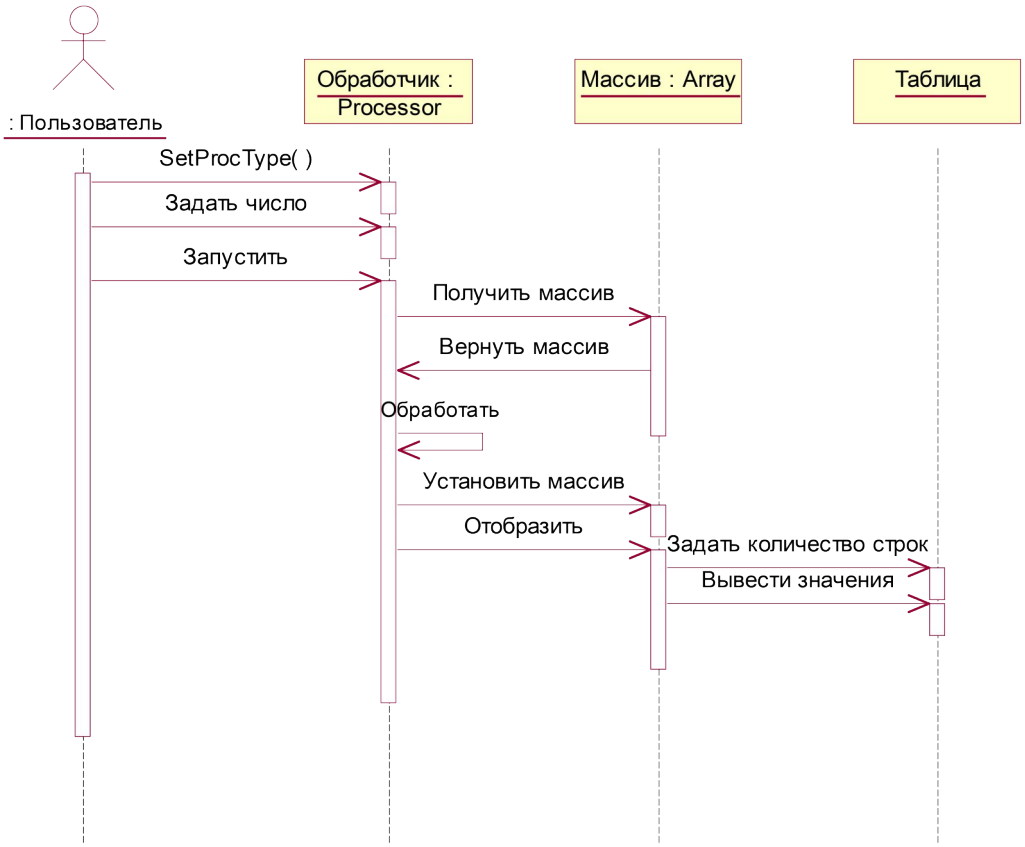


Рис. 25. Диаграмма последовательности с сообщением, замененным на операцию

ПРАКТИЧЕСКАЯ РАБОТА №4

РАЗРАБОТКА ДИАГРАММЫ КОМПОНЕНТОВ

Порядок выполнения работы

Диаграмма компонентов (Component Diagram) предназначена для отображения физической структуры системы в виде набора исходных и выполняемых файлов, контейнеров, подпрограмм, задач. Это делается для создания единого представления о том, какие подпрограммы и объекты будут сгруппированы в рамках тех или иных модулей (в частности, EXE, DLL, OCX и пр.). Эту группировку выполняют из соображений связности. Ее результаты используют, в частности, при назначении заданий проектировщикам.

Rational Rose 2000 позволяет разработать несколько диаграмм компонентов для отображения реализации с различной степенью детальности.

Для создания диаграммы компонентов можно воспользоваться пунктом главного меню BrowseàComponent Diagram… или запустить главную диаграмму компонентов из пункта Component View окна Browser.

Панель инструментов диаграммы компонентов показана на рис. 26.

Инструмент выбора (Selection Tool)
Текст (Text Box)
Комментарий (примечание, Note)
Якорь для замечания (Note Anchor)
Компонент (Component)
Контейнер (Package)
Зависимость (Dependency)
Спецификация подпрограммы (Subprogram Specification)
Тело подпрограммы (Subprogram Body)
Главная программа (Main Program)
Спецификация контейнера (Package Specification)
Тело контейнера (Package Body)
Спецификация задачи (Task Specification)
Тело задачи (Task Body)

Рис. 26. Панель инструментов для диаграммы компонентов

Особенность всех инструментов диаграммы компонентов в том, что все это элементы одного типа – компоненты, отличающиеся своим стереотипом. От стереотипа зависит, в частности, используемая компонентом пиктограмма. Поэтому если, например, разместить на диаграмме компонент (Component), а затем в диалоговом окне настройки его спецификации изменить его стереотип на Task Body, изображение его изменится.

Имя компонента совпадает с именем файла. Для компонента можно назначить язык программирования, что позволяет внести в проектирование специфику, ориентированную на конкретное средство разработки.

Инструмент Компонент (Component) предназначен для создания компонентов различного назначения, в первую очередь, компонентов периода выполнения ПС: файлов EXE, DLL, компонентов ActiveX.

Инструмент Контейнер (Package) позволяет изобразить в модели объединение компонентов. Помещение в контейнер компонента производится путем перетаскивания в окне Browser.

Зависимости (Dependency) позволяют изобразить направленные связи между компонентами. Зависимость прорисовывается в направлении от клиента к серверу.

Инструменты подпрограмм Subprogram Body и Subprogram Specification используются для

обозначения подпрограмм и объявлений подпрограмм.

Инструмент Главная программа (Main Program) используется не всегда. Например, во многих средах визуального программирования (Visual C++, Borland C++ Builder и т.п.) тело главной программы скрыто в библиотеках средства разработки, а разработчик работает с объектом-оберткой.

Инструменты контейнеров Package Body и Package Specification обычно служат контейнерами классов. Например, для языка C++ при помощи инструмента Package Body моделируется файл .cpp, содержащий один или несколько классов, а при помощи инструмента Package Specification – файл .h – заголовочной файл с объявлениями этих классов.

Инструменты Task Specification и Task Body позволяют моделировать независимые потоки в многопоточной системе.

Из приведенных характеристик видно, что можно формировать диаграммы компонентов различного рода: диаграммы файлов в процессе разработки (Package Body, Package Specification, Subprogram Body, Subprogram Specification, Main Program), диаграммы программных компонентов периода выполнения (Component, Task Body, Task Specification).

Сформируем диаграмму файлов исходного кода с ориентацией на язык C++.

Создадим заголовочный файл массива при помощи инструмента Package Specification. В диалоговом окне Component Specification назначим имя файла компонента Array и язык программирования C++ (рис. 27).

Если теперь нажать кнопку Применить (Apply), то вид диалогового окна изменится – в нем добавятся закладки, специфичные для C++ (рис. 28). В закладке Realizes следует указать классы, которые будут реализованы в данном контейнере. В нашем случае это класс Array. Для того чтобы указать на реализацию класса в данном модуле, нужно выполнить пункт Assign в контекстном меню соответствующего класса на закладке Realizes. После этого класс будет отмечен галочкой (рис. 28). Изменится также изображение класса в разделе Logical View окна Browser.

Создадим файл реализации массива при помощи инструмента Package Body. Здесь также необходимо назначить имя компонента (пусть называется также Array), назначить язык программирования и указать класс Array, который будет реализован в этом компоненте.

Теперь следует указать связь в направлении от тела контейнера к спецификации контейнера (рис. 29). Это делается для того, чтобы в исходный текст файла .CPP была вставлена директива включения #include ”Array.h”.

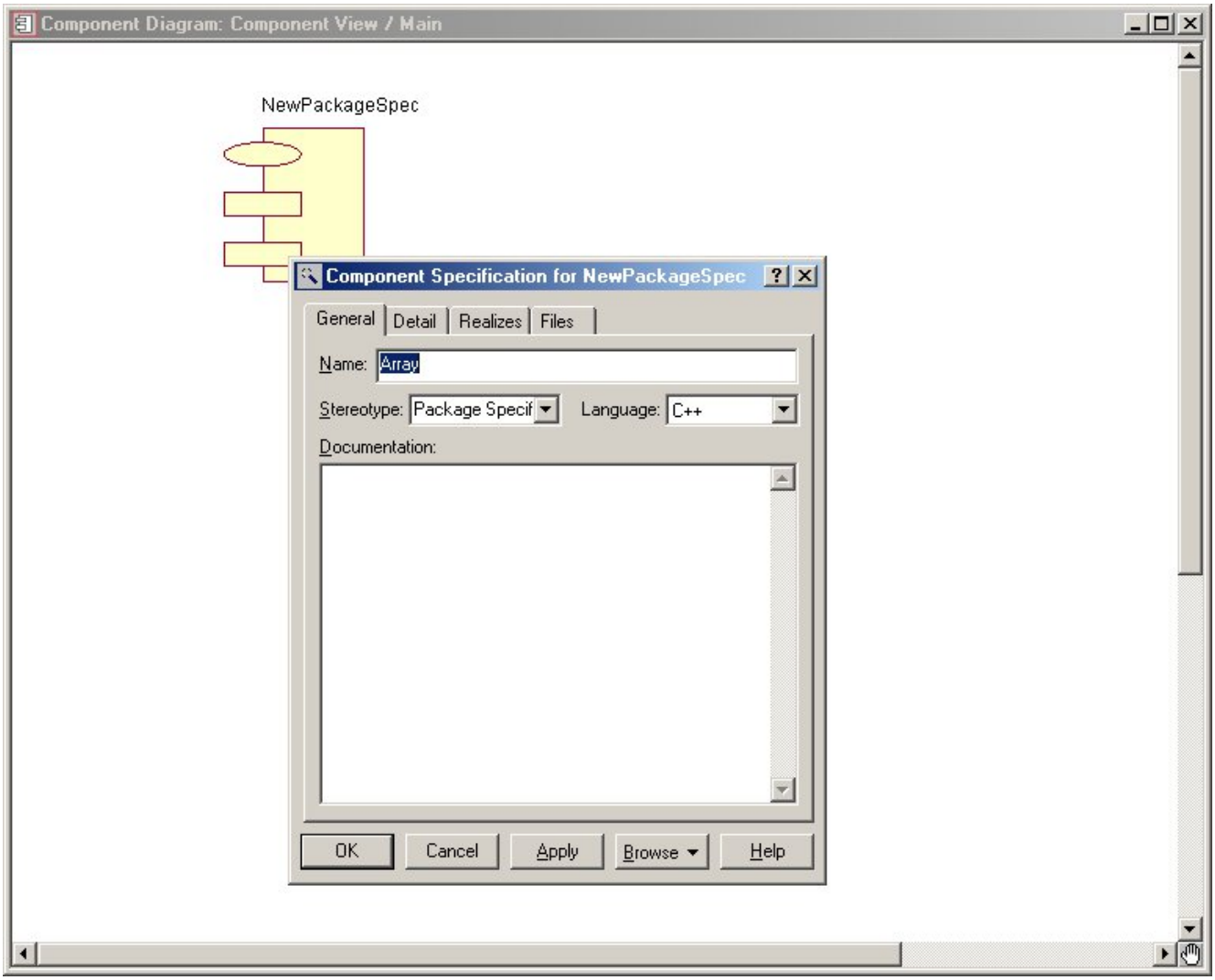


Рис. 27. Создание и настройка новой спецификации контейнера

Аналогично описанному создадим набор из спецификаций и тел контейнеров для всех объектов. Установим связи между ними таким образом, чтобы спецификация контейнера класса Array «была видна» спецификации класса Processor. В этом случае Обработчик сможет обращаться к операциям класса Массив. Получим диаграмму рис. 30.

Диаграмма компонентов позволяет получать заготовки исходного кода. Генерация кода выполняется при помощи пункта контекстного меню компонента C++àCode Generation. При этом создаются файлы .CPP и .H, содержащие реализации и объявления классов соответственно. Здесь автоматически реализуются заготовки конструкторов и переопределение некоторых операторов. При групповой разработке проекта эти заготовки можно распределить между разработчиками для дальнейшей реализации.

При генерации кода файлы .CPP и .H создаются в директории C:\Program Files\Rational\Rose 2000\c++\source.

Генерация кода на этой стадии разработки должна выдать множество предупреждений и сообщений об ошибках. Это объясняется тем, что структура классов не была доработана. После создания компонентов и указания языка программирования следует доработать структуры классов, указать типы свойств, операций, атрибутов операций, при необходимости – подкорректировать классы видимости.

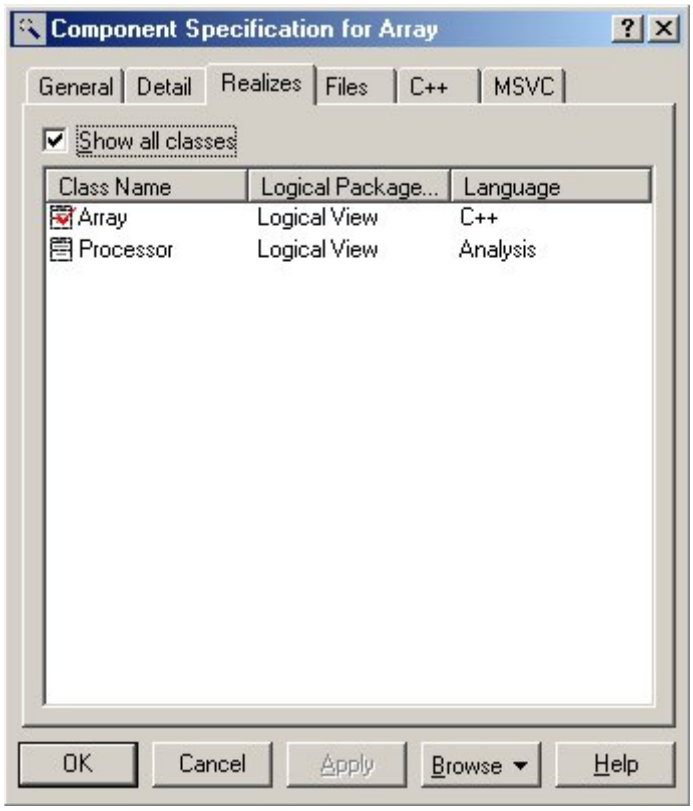


Рис. 28. Классы, реализуемые в компоненте

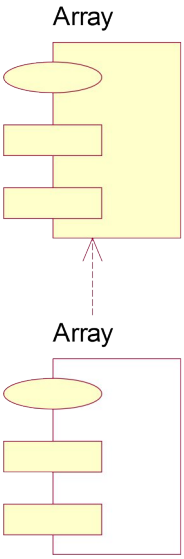


Рис. 29. Организация связи между спецификацией контейнера и телом контейнера

Во избежание затирания впоследствии чужими файлами свои следует перенести в свой каталог.

Выполнение работы следует завершить получением полного набора исходных файлов проекта и продемонстрировать преподавателю результаты генерации кода. Следует добиться полного отсутствия сообщений об ошибках и предупреждений.

Внимательно изучите сгенерированный код. Проанализируйте, как полученные модели были реализованы в полученном коде. Обратите внимание на использование в полученных файлах директив #include, private, public, protected. Ответьте на приведенные ниже контрольные вопросы.

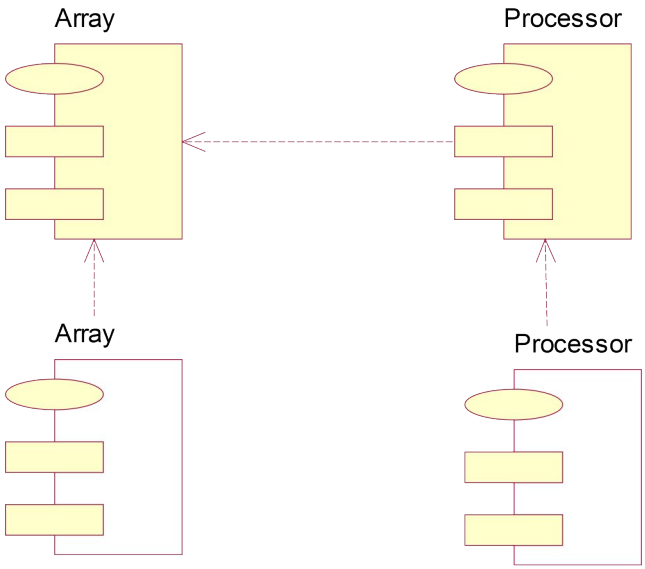


Рис. 30. Диаграмма компонентов

Контрольные вопросы

1. Как производится настройка класса на язык программирования?
2. Как производится связывание класса с модулем?
3. Как настраивается тип операции (свойства) класса?
4. Как создаются атрибуты операции класса, и как они выглядят в сгенерированном коде?
5. Как настраиваются классы видимости свойств и операций класса, и как они указываются в сгенерированном коде?
6. Какие операции класса, за исключением объявленных разработчиком, создаются при генерации кода автоматически?
7. Как при генерации кода обрабатываются объявленные отношения агрегации?
8. Как обрабатываются при генерации кода отношения зависимости между компонентами?
9. Что необходимо для организации чтения/записи приватных свойств класса, и как эта функциональность поддерживается Rational Rose?

ПРАКТИЧЕСКАЯ РАБОТА №5

РАЗРАБОТКА ДИАГРАММЫ СХЕМ СОСТОЯНИЙ (STATECHART)

Порядок выполнения работы

Диаграммы активности и схем состояний не являются компонентами модели, используемыми при генерации кода. Они используются для моделирования поведения классов и являются важными компонентами проектной документации.

Диаграммы состояний предназначены для независимого описания поведения классов как конечных автоматов. При этом поведение описывается как совокупность состояний, переходы между которыми выполняются при выполнении определенных условий и сопровождаются некоторыми действиями.

Создание диаграммы состояний выполняется несколькими способами.

Способ 1. В окне Browser в контекстном меню папки Logical View следует выбрать NewàStatechart Diagram, а затем ввести имя новой диаграммы.

Способ 2. В главном меню программы следует выбрать BrowseàState Machine Diagram…, после чего выбрать пункт <New> (или выбрать имя диаграммы при необходимости редактирования созданной ранее), а затем в диалоговом окне (рис. 31) задать имя и тип диаграммы. Это диалоговое окно является общим для обоих типов диаграмм автоматов – и для диаграмм схем состояний, и для диаграмм активности, поэтому в данном случае необходимо

выбрать тип Statechart для разработки диаграммы схем состояний.

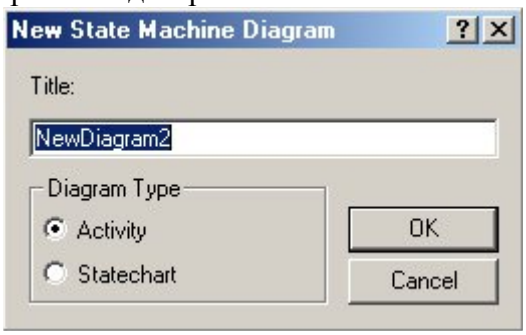


Рис. 31. Диалоговое окно ввода имени и типа диаграммы автоматов

Инструмент выбора (Selection Tool)
Текст (Text Box)
Комментарий (примечание, Note)
Якорь для замечания (Note Anchor)
Состояние (State)
Начало (Start State)
Конец (End State)
Переход (State Transition)
Переход в то же состояние (Transition to Self)

Рис. 32. Панель инструментов для диаграммы схем состояний

Способ 3. В диаграмме классов из контекстного меню класса выбрать пункт Sub DiagramsàNew Statechart Diagram. После этого появится окно Statechart Diagram. Созданная диаграмма будет привязана к классу, поведение которого собирался моделировать разработчик. Таким образом, этот способ создания диаграммы схем состояний является наиболее естественным. Диаграмму в дальнейшем можно переименовать из окна Browser.

Созданные диаграммы доступны из окна Browser. Панель инструментов диаграммы схем состояний показана на рис. 32.

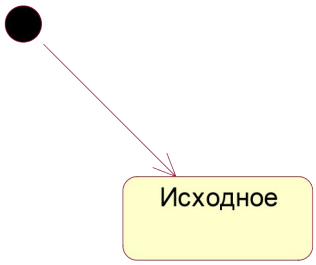


Рис. 33. Схема, содержащая начальное состояние, переход и исходное состояние

Сразу выделим основные состояния, в которых должен пребывать Массив:

- исходное состояние;
- состояние «изменен»;
- состояние «сохранен».

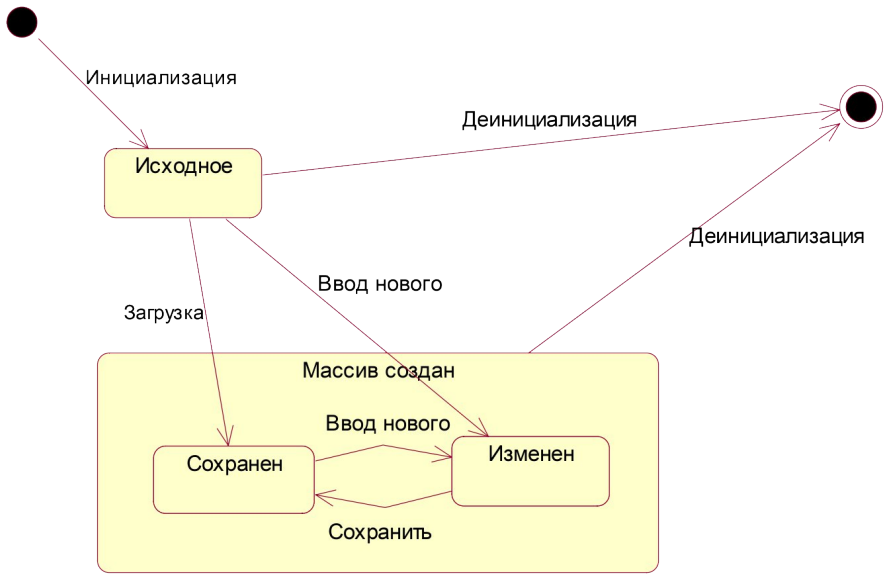


Рис. 34. Общая схема состояний главного контроллера

Переход в исходное состояние осуществляется из начального состояния после инициализации объекта Массив. Переход из любого из состояний в конечное состояние осуществляется в результате деинициализации объекта. Переход из исходного состояния в состояние «сохранен» осуществляется в результате загрузки массива из файла. Переход из исходного состояния в состояние «изменен» осуществляется в результате ручного ввода массива. Переход из состояния «изменен» в состояние «сохранен» выполняется в результате выполнения операции сохранения массива. Переход из состояния «сохранен» в состояние «изменен» выполняется в результате ручного ввода нового массива.

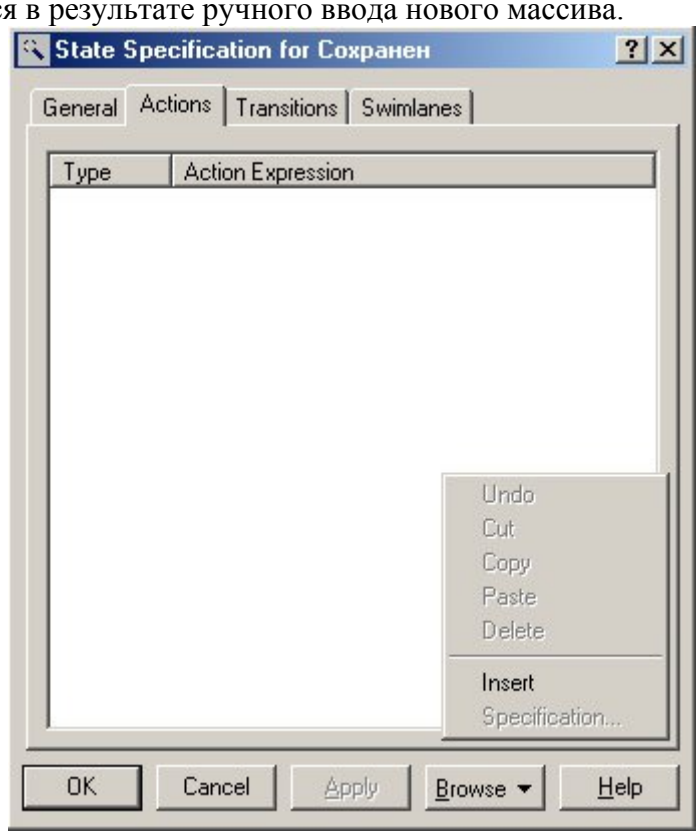


Рис. 35. Диалоговое окно спецификации состояния

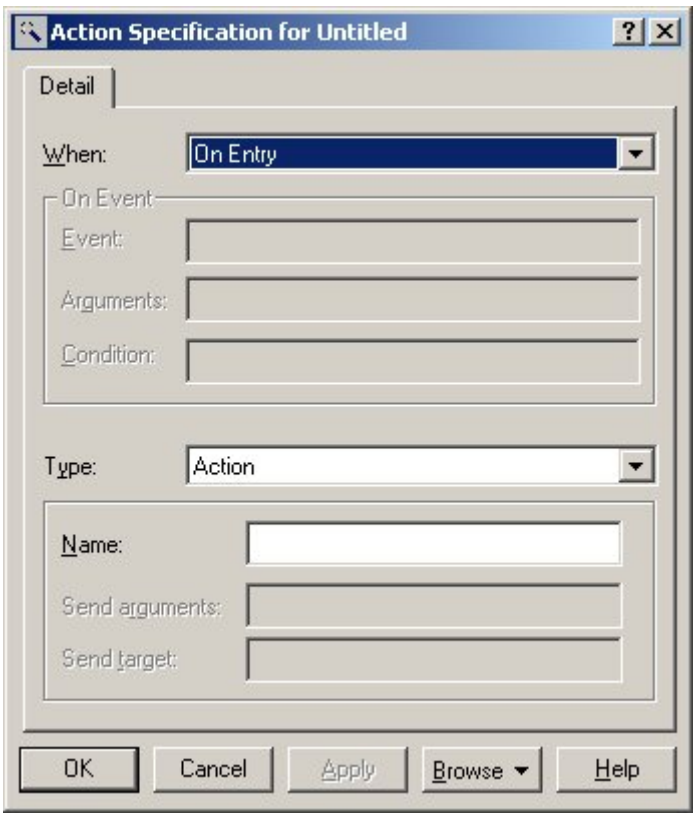


Рис. 36. Диалоговое окно спецификации действия

Можно также объединить состояния «сохранен» и «изменен» в надсостояние «создан».

Размещение состояний на диаграмме выполняют в интуитивно очевидном порядке. Сначала размещают начальное состояние. Можно задать его имя или затереть предлагаемое имя и оставить его безымянным, т.к. пиктограмма указывает на назначение этого состояния. Затем разместим исходное состояние и свяжем эти два состояния переходом. Спецификация перехода настраивается в диалоговом окне, вызываемом двойным щелчком на переходе или через контекстное меню перехода. В спецификации можно ввести имя события, вызвавшего переход, а также сторожевое условие перехода, посылаемые сообщения и т.д. В данном случае ограничимся именем события. Получим схему, показанную на рис. 33.

Добавим надсостояние «создан» и разместим в нем состояния «сохранен» и «изменен». Организуем переходы между состояниями (рис. 34).

Таким образом, общая схема состояний получена. Следует ее детализировать. В ходе детализации следует указать действия, связанные с состояниями.

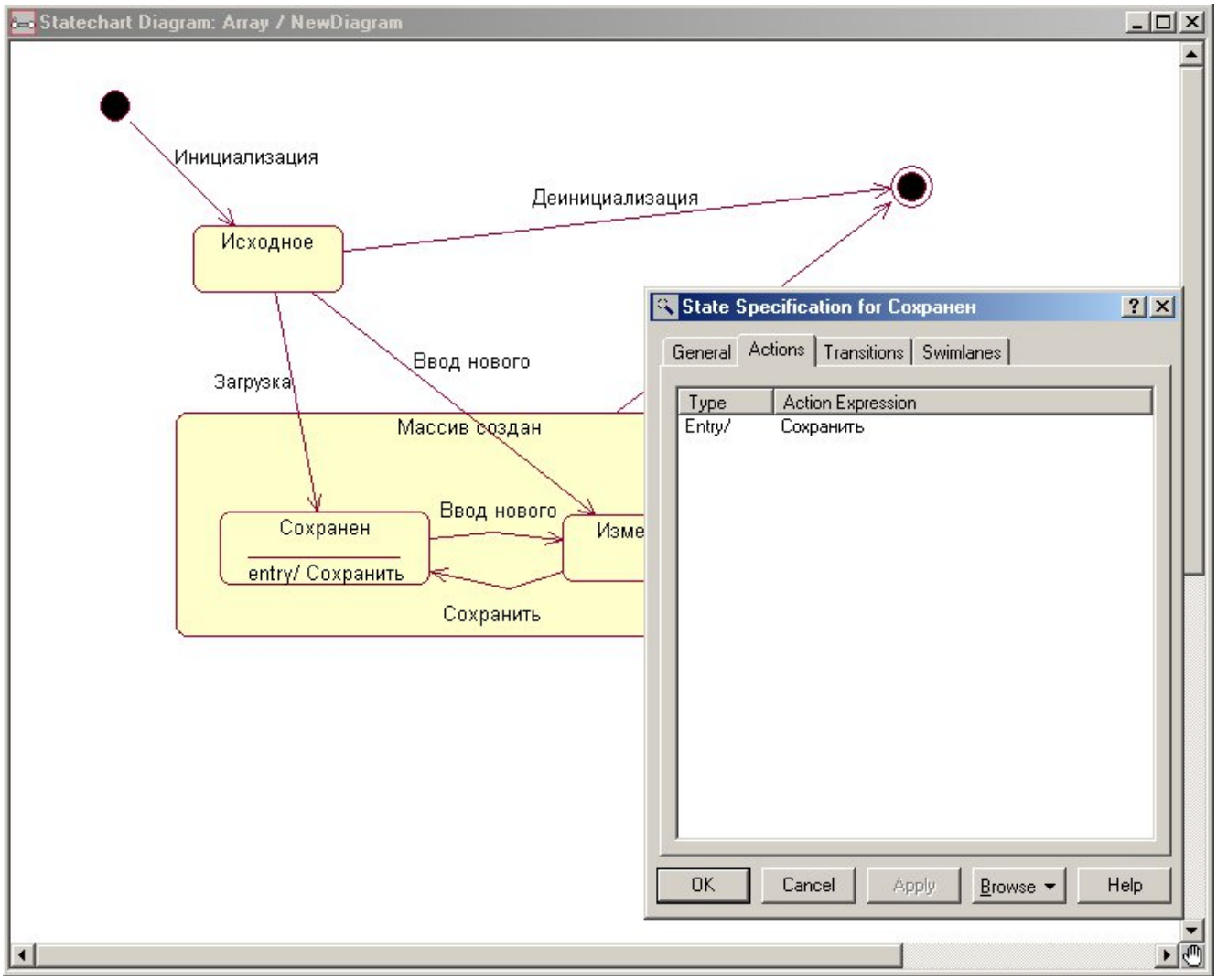


Рис. 37. Состояние с указанными действиями

Для настройки состояния «сохранен» следует в его контекстном меню выбрать пункт Open Specification…, а в появившемся диалоговом окне вкладку Actions. При выборе пункта Insert контекстного меню таблицы, размещенной во вкладке Actions (рис. 35), создается новое действие, обозначаемое меткой типа Entry/ в таблице. Для редактирования действия следует выполнить двойной щелчок мышкой на этой метке.

Типы меток действий:

On Entry – действие выполняется при входе в состояние;

On Exit – действие выполняется при выходе из состояния;

Do – действие выполняется в процессе нахождения в состоянии;

On Event – действие выполняется в случае наступления определенного события.

В нашем случае имеет смысл обозначить действие «Сохранить» по входу в состояние «сохранен». Поэтому в нижней части окна рис. 36 выбираем тип Action, вводим имя Сохранить. Список действий в окне спецификации состояния и новый вид диаграммы состояний показаны на рис. 37.

При выполнении работы необходимо, основываясь на материале предыдущих лабораторных работ, изобразить диаграммы схем состояний всех классов.

ПРАКТИЧЕСКАЯ РАБОТА №6

РАЗРАБОТКА ДИАГРАММЫ АКТИВНОСТИ (ACTIVITY)

Порядок выполнения работы

Диаграммы активности предназначены для отображения взаимосвязанных

последовательностей действий, производимых объектом в течение его существования.

Создание диаграммы активности выполняется несколькими способами.

Способ 1. В окне Browser в контекстном меню папки Logical View следует выбрать NewàActivity Diagram, а затем ввести имя новой диаграммы.

Инструмент выбора (Selection Tool)
Текст (Text Box)
Комментарий (примечание, Note)
Якорь для замечания (Note Anchor)
Состояние (State)
Действие (Activity)
Начало (Start State)
Конец (End State)
Переход (State Transition)
Переход в то же состояние (Transition to Self)
Горизонтальная синхронизация (Horizontal Synchronization)
Вертикальная синхронизация (Vertical Synchronization)
Условное ветвление (Decision)
«Плавательная дорожка» (Swimlane)

Рис. 38. Панель инструментов для диаграммы активности

Способ 2. В главном меню программы следует выбрать BrowseàState Machine Diagram…, после чего выбрать пункт <New> (или выбрать имя диаграммы при необходимости редактирования созданной ранее), а затем в диалоговом окне (рис. 31) задать имя и тип диаграммы. Это диалоговое окно является общим для обоих типов диаграмм автоматов – и для диаграмм схем состояний, и для диаграмм активности, поэтому в данном случае необходимо выбрать тип Activity для разработки диаграммы активности.

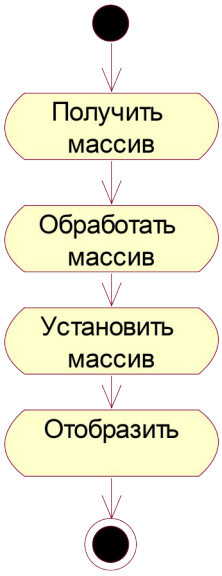


Рис. 39. Диаграмма активности

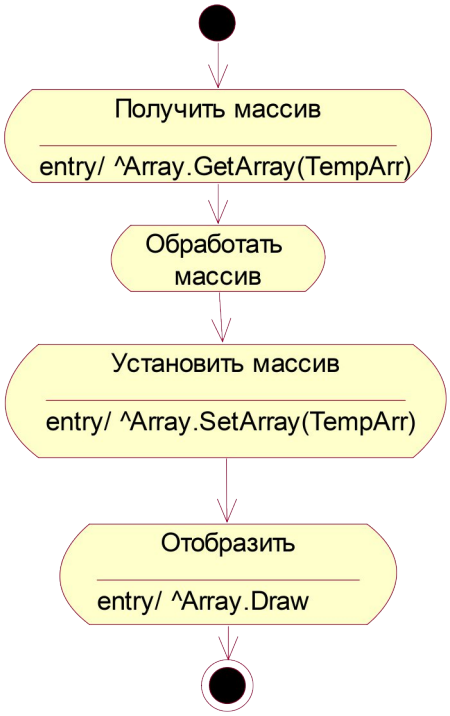


Рис. 40. Диаграмма активности

Способ 3. В диаграмме классов из контекстного меню класса выбрать пункт Sub DiagramsàNew Activity Diagram. После этого появится окно Activity Diagram. Созданная диаграмма будет привязана к классу, поведение которого собирался моделировать разработчик. Таким образом, этот способ создания диаграммы активности является наиболее естественным. Диаграмму в дальнейшем можно переименовать из окна Browser.

Созданные диаграммы доступны из окна Browser. Панель инструментов диаграммы активности показана на рис. 38.

Как можно видеть по рис. 38, инструментарий диаграмм активности отличается от классического, в частности, кроме активности, в диаграммах могут использоваться состояния.

Во многих случаях различия между состояниями и действиями достаточно условны, и диаграммы активности оказываются с точностью до обозначений похожи на диаграммы состояний.

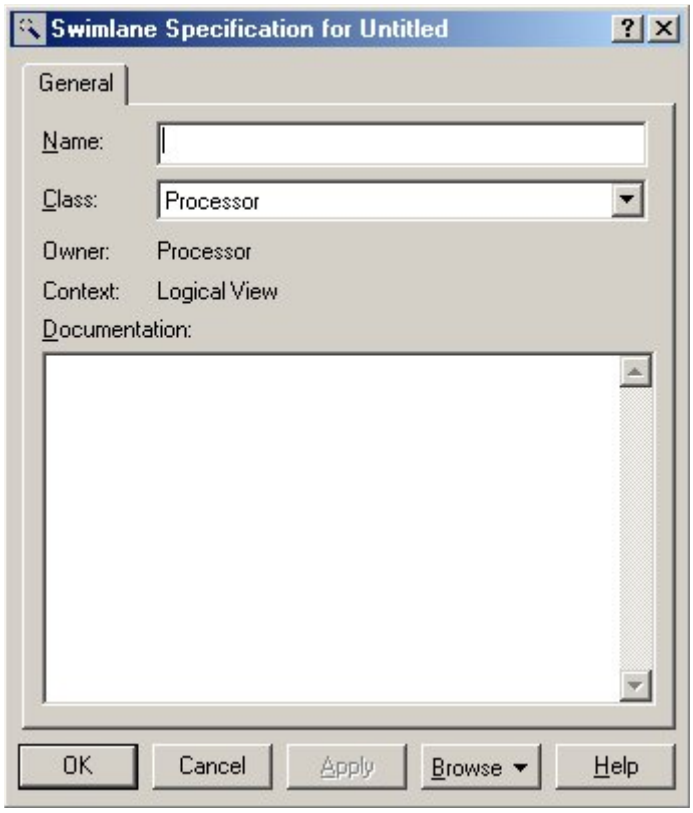


Рис. 41. Настройка плавательной дорожки

В рассматриваемом примере имеет смысл построить диаграмму активности процесса обработки массива обработчиком. В данном случае диаграмма активности является средством формализации алгоритма обработки и формирования проектного документа, который формируется проектировщиком и передается разработчику (программисту).

Как было рассмотрено в диаграмме последовательности (рис. 25), основными действиями обработчика при обработке массива должны быть: получение массива у объекта класса Array, обработка полученного массива, установка массива, т.е. передача обработанного массива объекту класса Array, и выдача этому объекту команды отображения массива. Сформированная таким образом диаграмма приведена на рис. 39.

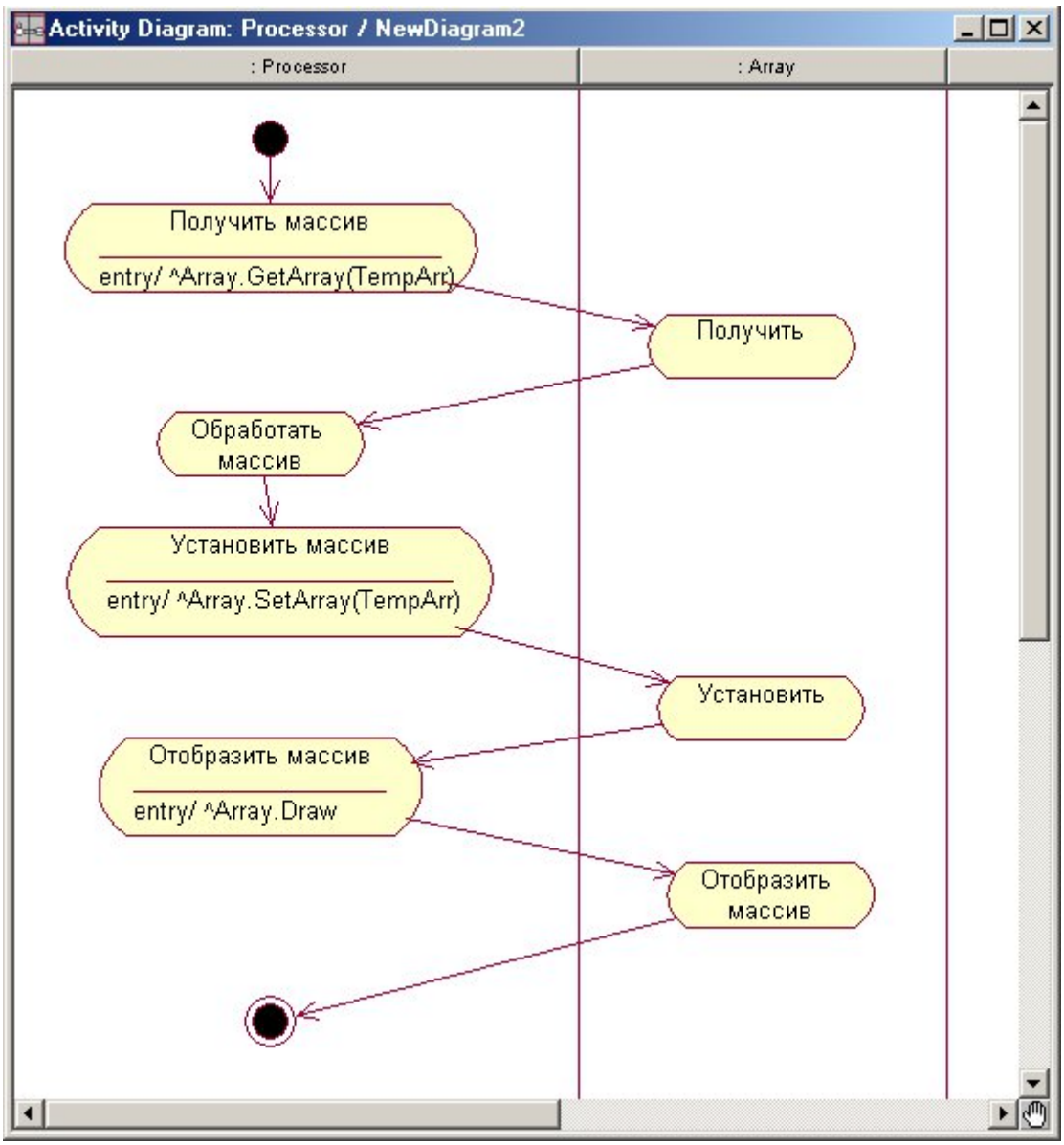


Рис. 42. Диаграмма активности с плавательными дорожками

Действия Получить, Установить и Отобразить связаны с обращением к операциям класса Array. Для получения массива операции GetArray класса Array передается адрес временного массива. Этот временный массив обрабатывается, затем его же адрес передается операции SetArray класса Array. Произведем настройку действий для конкретизации этих решений. Настройка действий выполняется аналогично настройке состояний, описанной выше (рис. 35, 36). В данном случае при входе в действие (On Entry) необходимо выполнить посылку сообщения объекту Array. Таким образом, при настройке действия Получить массив нужно в нижней части диалогового окна рис. 36 выбрать тип Send Event, указать имя сообщения (Name) GetArray, указать аргумент (Send arguments) TempArr (временный массив), указать объект, которому посылается сообщение, (Send target) Array. Диаграмма приобретает вид рис. 40.

Для того чтобы отметить взаимодействие экземпляра класса Processor с экземпляром класса Array, используем плавательные дорожки. Создадим две плавательные дорожки и настроим их через их спецификации (в частности, укажем классы, к которым они относятся – рис. 41). В данном случае детализация с указанием операций класса Array уже была произведена, поэтому детализировать действия, выполняемые классом Array, не будем – достаточно указания на взаимодействие с ним объекта класса Processor (рис. 42).

Действие Обработать массив требует более серьезного алгоритмического рассмотрения. Поэтому создадим для этого действия вложенную диаграмму активности при помощи пункта контекстного меню Sub DiagramsàNew Activity Diagram.

Обработку будем производить в цикле путем умножения каждого элемента массива на число, которое в зависимости от типа обработки проинициализируем либо заданной величиной, либо величиной, обратной заданной. Обработка должна содержать в себе: инициализацию цикла, инициализацию множителя, обработку элемента массива, инкремент счетчика цикла и элементы условного ветвления. В общих чертах диаграмма активности имеет вид рис. 43. Далее можно уточнять ее вплоть до разработки фрагментов кода (рис. 44).

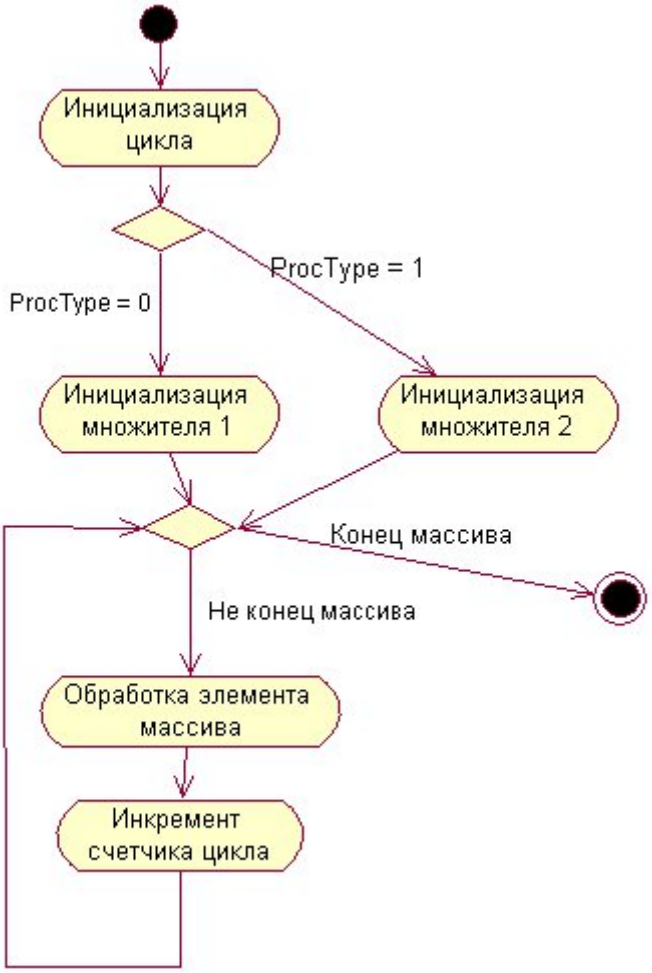


Рис. 43. Диаграмма активности обработки массива

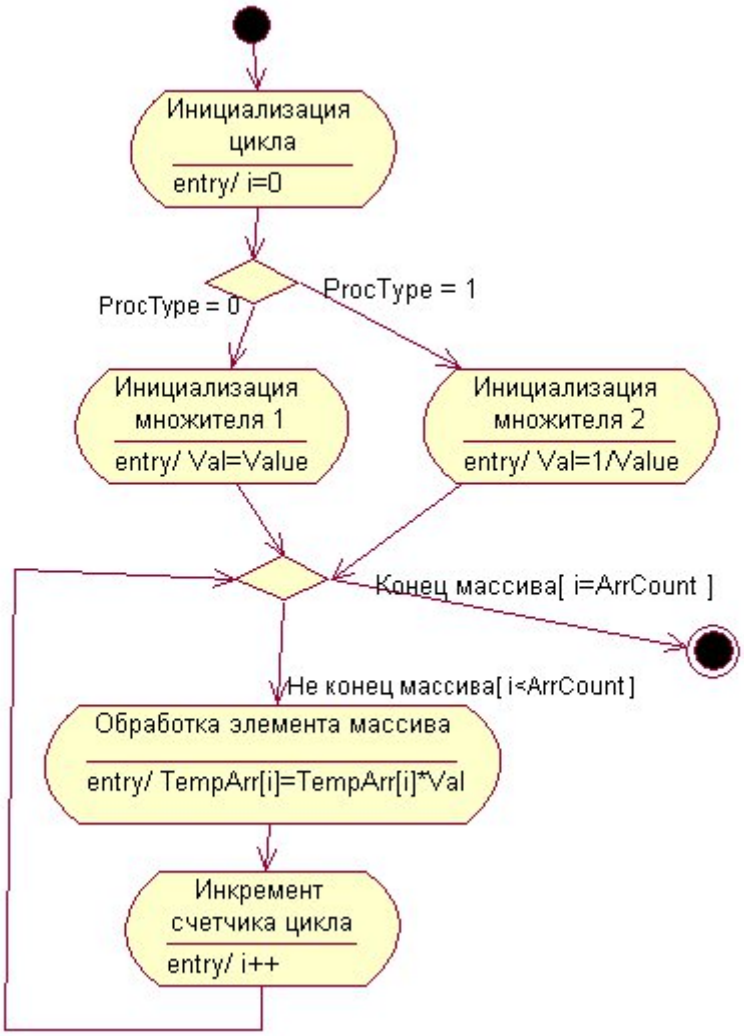


Рис. 44. Конкретизированная диаграмма активности обработки массива

ПРАКТИЧЕСКАЯ РАБОТА №7

ОБРАТНЫЙ ИНЖИНИРИНГ

Порядок выполнения работы

В терминологии Rational Rose обратным инжинирингом называется процесс восстановления модели по исходному коду на языке программирования.

При ориентации на язык C++ обратная инженерия выполняется при помощи средства Rational Rose C++ Analyzer. Он может быть запущен как из пакета программ Rational Rose 2000 Enterprise Edition, так и при помощи пункта главного меню Rational Rose 2000 ToolsàC++àReverse Engineering.

Процесс обратного инжиниринга начинается с создания проекта в среде C++ Analyzer. Создание проекта выполняется при помощи пункта главного меню FileàNew или соответствующей кнопки панели инструментов.

Главное окно редактирования проекта показано на рис. 45.

Кнопка Caption… предназначена для задания наименования проекта. Наименование проекта вводится в диалоговом окне рис. 46.

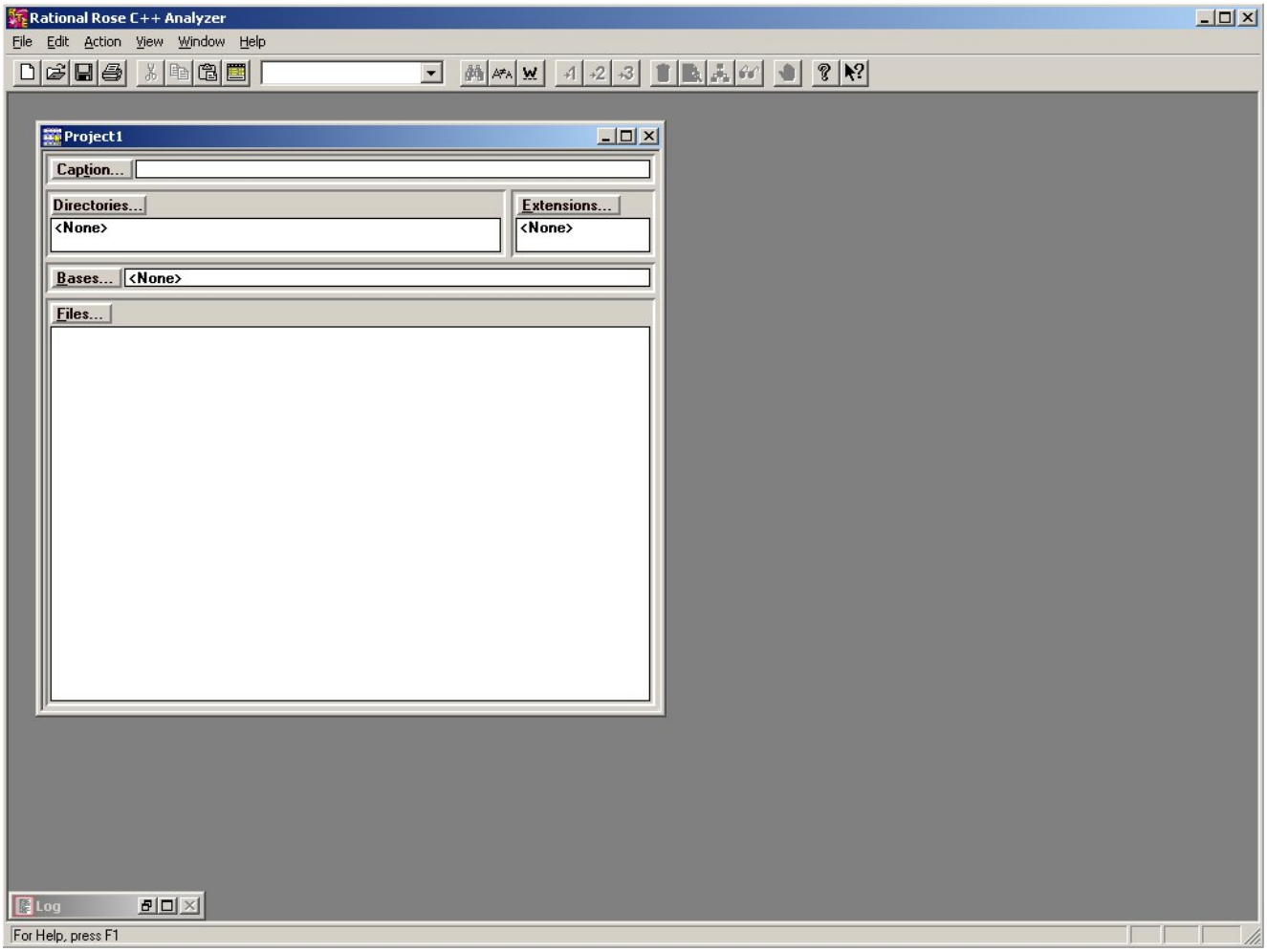


Рис. 45. Общий вид среды Rational Rose C++ Analyzer и главное окно редактирования проекта

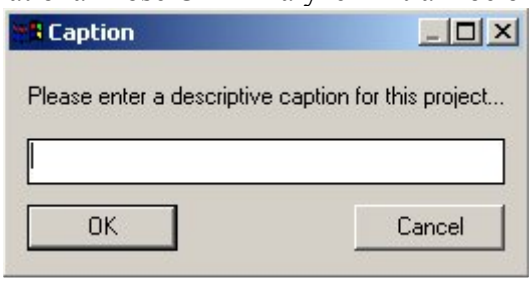


Рис. 46. Диалоговое окно ввода наименования проекта

Список директорий проекта формируется в диалоговом окне рис. 47, вызываемом при помощи кнопки Directories…. Для больших и разветвленных проектов полезна возможность формирования списка директорий. Набор элементов управления Current Directory Name позволяет выбрать диск (Drives), подключить сетевой диск (Network…), выбрать директорию (Directory Structure), добавит в список текущую директорию (Add Current), ее поддиректории (Add Subdirs), или вложенную иерархию (Add Hierarchy).

Очевидно, в простейшем случае, когда все файлы исходных кодов (*.cpp, *.h) размещаются в одной директории достаточно найти ее в дереве и нажать кнопку Add Current. При этом выбранная директория появляется в списке Source Directory – Data Directory. Здесь в колонке Source Directory отображается выбранная директория исходных кодов, а в колонке Data Directory – директория, где будут храниться служебные файлы проекта.

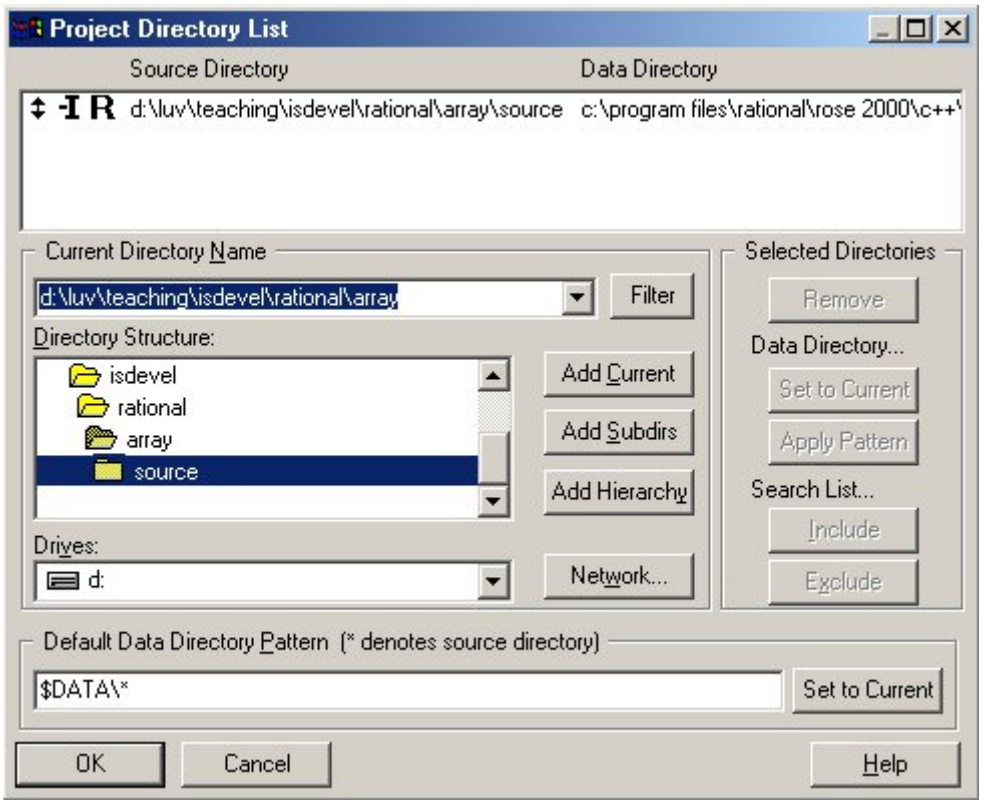


Рис. 47. Диалоговое окно списка директорий проекта

Пиктограмма ↕ предназначена для изменения положения директории в списке. Пиктограмма I предназначена для указания, включается ли директория в поиск заголовочных файлов, подключаемых директивами #include. При помощи пиктограммы R в создаваемой после обратного инжиниринга модели можно установить запрет на генерацию кода, чтобы не повредить исправления, внесенные в исходный код после его генерации.

При помощи кнопки Extensions… выводится диалоговое окно рис. 48, где можно указать расширения (в нашем случае *.h и *.cpp), использованные в проекте. C++ Analyzer установит соответствующие им расширения свойих рабочих файлов.

Кнопка Bases… предназначена для указания базового проекта, полученного при помощи C++ Analyzer ранее. Это бывает необходимо, когда в крупном проекте некоторый его модуль подключается к остальной части в виде библиотеки, а для успешной компиляции остальными частями проекта используется заголовочный файл с объявлениями функций и классов библиотеки. В этом случае создают проект по библиотеке, а затем его включают в качестве базового во все проекты, где эта библиотека используется. Это справедливо и для стандартных библиотек средства разработки. В данном случае не будем формировать список базовых проектов.

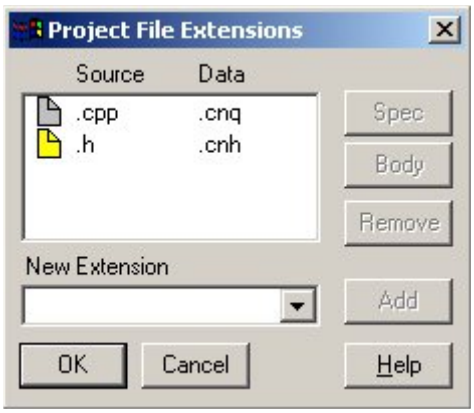


Рис. 48. Диалоговое окно формирования списка расширений проекта

При помощи кнопки Files… можно указать конкретные файлы, которые предстоит анализировать и реконструировать по ним модели. Диалоговое окно выбора файлов приведено

на рис. 49. В списке Files Not In List указаны файлы, входящие в отмеченную ранее директорию, в списке Files In List – файлы, входящие в список анализируемых. Из первого списка нужно переместить необходимые файлы во второй. Это делается при помощи кнопок Add Selected, Add All. Выберем все файлы, полученные ранее генерацией кода. Примерный вид окна проекта (для примера, рассмотренного в материале предыдущих работ) показан на рис. 50.

Для выполнения дальнейших действий нужно отметить все файлы проекта, как показано на рисунке (например, при помощи клавиши Shift и мыши).

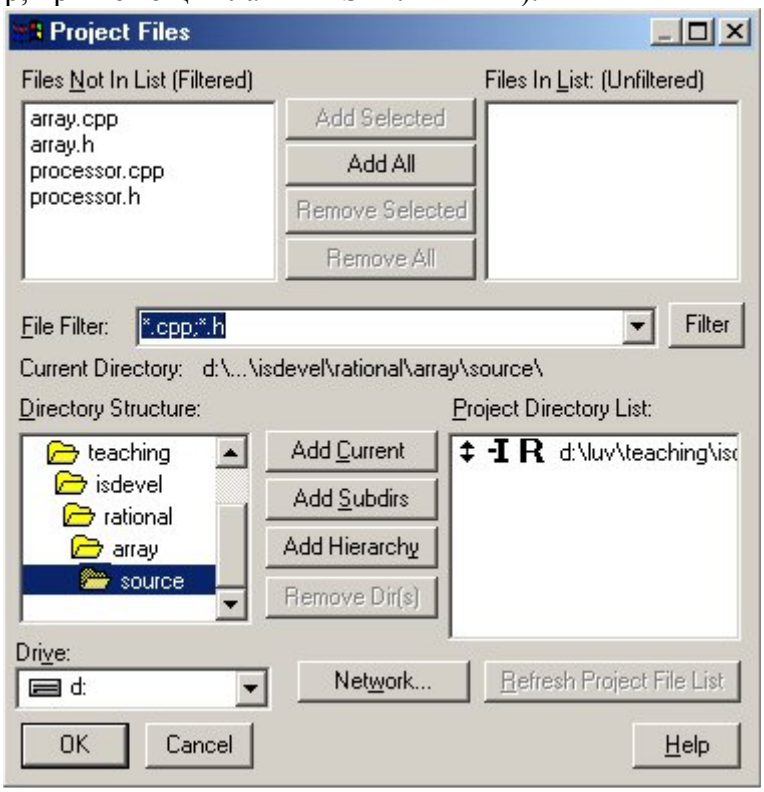


Рис. 49. Диалоговое окно формирования списка файлов проекта

Далее производится анализ при помощи пункта главного меню ActionàAnalyze. В результате анализа создаются рабочие файлы проекта в директории, указанной во втором столбце списка Directories. Обратите внимание на созданные файлы, их расширения и содержимое.

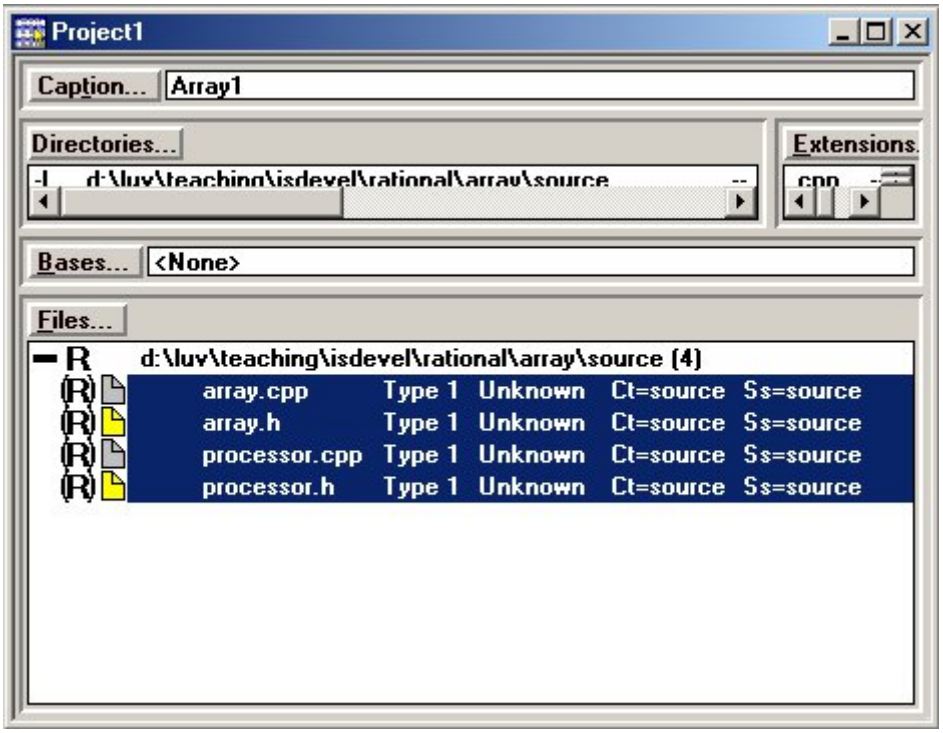


Рис. 50. Главное окно редактирования проекта после формирования списка файлов

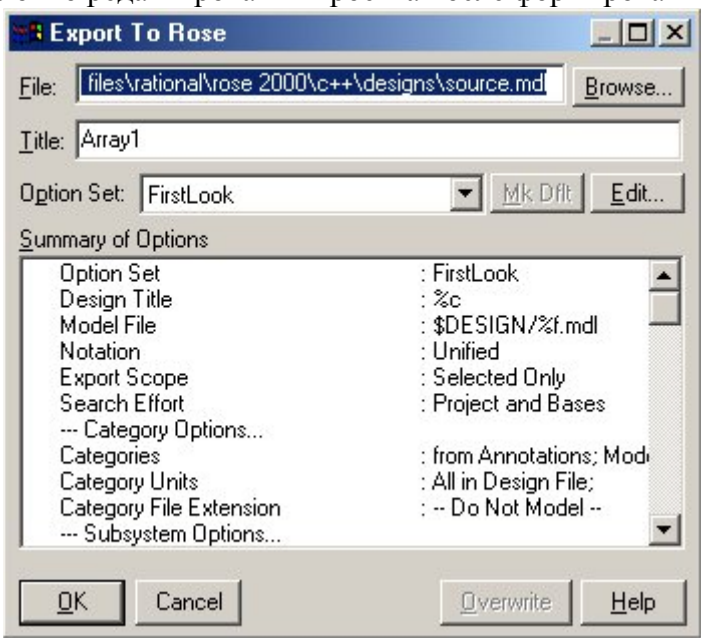


Рис. 51. Диалоговое окно запуска экспорта модели в Rose

После проведения анализа можно запускать экспорт модели при помощи пункта главного меню ActionàExport to Rose… (рис. 51). При помощи поля ввода File и кнопки Browse… можно изменить имя файла, предлагаемое по умолчанию. Чтобы не потерять диаграммы, полученные в предыдущих лабораторных работах, не указывайте имя файла модели, в котором выполнялись предыдущие работы. При помощи списка выбора Option Set и кнопки Edit… можно изменять настройки экспорта (в настоящей работе не рассматривается). Кнопкой OK запускается процесс сохранения модели.

Полученный файл может быть открыт Rational Rose 2000.

При выполнении настоящей работы следует путем выполнения описанных действий восстановить модель с диаграммой классов из кода, сгенерированного в ходе выполнения предыдущей практической работы.

Убедившись в корректном восстановлении диаграммы классов, следует исследовать зависимость работы анализатора кода от служебных комментариев, расставляемых Rational

Rose в генерируемом коде. Как восстанавливается класс, если в соответствующем модуле удалить комментарии внутри объявления класса? Как восстанавливается класс, если удалить комментарии вне объявления класса?

9.2 Методические рекомендации по подготовке письменных работ

Порядок составления и оформления отчета о практической работе

В значительной мере эффективность решения задачи по выполнению практической работы зависит от качества соответствующего отчета. Для этого необходимо соблюдать следующие основные требования по составлению и оформлению отчета, обусловленные соответствующими нормативными документами. Текст отчета должен быть лаконичным и вместе с тем информативным. Текст должен быть изложен с соблюдением правил грамматики. Отчет составляется с обязательным составлением следующих разделов:

1. Заголовок отчета.
2. Цели работы.
3. Методика работы.
4. Порядок выполнения работы (этапы работы).
5. Выводы по работе.

1. В **заголовке отчета** приводятся наименования идентифицирующих признаков: **Отчет о практической работе № 1** по теме, например, «Use-case диаграммы», ниже указываются данные студента (фамилия и инициалы, вид обучения, специальность, курс, группа).

2. В разделе **Цель работы** формулируется цели работы студента в соответствии с содержанием раздела «Постановка задачи» данной работы и индивидуального задания студенту на работу.

3. В разделе **Методика работы** указывается методика работы в соответствии с имеющейся формулировкой в разделе «Методика работы» данной работы и при необходимости уточняется в зависимости от содержания конкретного варианта задания студенту на практическую работу.

4. **Порядок выполнения работы**. Приводятся номера и наименования этапов работы, предусмотренные для работы данного Практикума. По каждому из этапов приводится описание выполненных студентом работ, направленных на достижение цели работы. Пропуск какого-либо из этапов работы Практикума не допускается. В рамках этапов помещается соответствующий иллюстративный материал - таблицы, рисунки (графики), полученные по ходу решения задачи работы. Обозначение иллюстративного материала выполняется в соответствии с правилами, принятыми для публикаций. Обозначение каждой таблицы и рисунка должно иметь номер и наименование. Внутри каждого отчета таблицы и рисунки обозначаются соответственно сквозными номерами. Обозначение таблицы указывается над таблицей, а обозначение рисунка под рисунком. Приводимые в тексте данной работы примеры включать в отчет не разрешается. Применяется только материал, полученный в ходе работы студентом по соответствующему заданию, полученному от преподавателя.

5. Последним разделом отчета являются **выводы** по работе. Это самая сложная и трудная часть работы. Очень важно, чтобы выводы отражали методику, технологию, применяемые программно-аппаратные средства решения задачи. Полезно каждому из этапов работы формулировать не менее одного вывода. Вывод может содержать от одного до трех предложений. Формулировки выводов должны быть конкретными, информативными, лаконичными, по возможности подкрепляться количественными данными.

Оформление отчета выполняется с учетом общепринятых правил. Графическая часть отчетов должна соответствовать правилам графического оформления. Текст отчета набирается в редакторе Word через 1,5 интервала, 14 кегль. Следует использовать шрифт Times New Roman. Заголовки разделов и подразделов выделяются жирным шрифтом. После окончания оформления

отчета он проверяется студентом на предмет качество содержания и формы. При условии обнаружения ошибок последние исправляются. После устранения дефектов отчета его экранная форма, или принтерная распечатка предъявляется преподавателю. При условии обнаружения преподавателем ошибок в отчете студент их исправляет и предъявляет отчет преподавателю повторно. Если ошибок нет, то отчет принимается и сохраняется на жестком диске.

Отчет по работе сохраняется студентом в виде отдельного файла. В имени файла указывается фамилия студента и номер выполненной работы. Файл сохраняется в папке с фамилией студента в папке соответствующей студенческой группы. Папка группы создается на первом занятии. В имени папки группы должен присутствовать индекс группы. Папка группы включается в папку «Мои документы».

Приложение 1. Аннотация
рабочей программы дисциплины

АННОТАЦИЯ РАБОЧЕЙ ПРОГРАММЫ ДИСЦИПЛИНЫ

Цель дисциплины: приобретение знаний, навыков и умений в области применения современных подходов к проектированию, разработке, тестированию и эксплуатации программных продуктов.

Задачи:

- изучение и сравнительный анализ современных процессов проектирования и разработки программных продуктов;
- изучение принципов и методов оценки качества и управления качеством программного продукта;
- приобретение практических навыков формирования и анализа требований, оценки качества и тестирования программных продуктов.

Знать: современные процессы проектирования и разработки программных продуктов; принципы управления качеством программного обеспечения; методы тестирования программного продукта.

Уметь: проводить сравнительный анализ процессов проектирования и разработки программных продуктов и делать обоснованный выбор; выполнять формирование и анализ требований для разработки программных продуктов; разрабатывать документацию, необходимую для тестирования программного продукта; выполнять тестирование программного продукта.

Владеть: информацией о процессах разработки и жизненном цикле программного обеспечения; инструментарием для разработки и тестирования программных продуктов.