Глава 9. Методические материалы по дисциплинам

9.1. Инжиниринг ПО

Дисциплина ориентирована на дальнейшее развитие навыков программирования и проектирования ПО с использованием шаблонов проектирования и стандартных технологических средств языка программирования и среды разработки, получение навыков написания качественного управляемого кода в проектах небольшого и среднего размера.

Лабораторный практикум

Лабораторная работа 1. Графический интерфейс с динамической разметкой. Разработать графический интерфейс с программируемой разметкой: переменное количество элементов, их расположение и перемещение, регулярная обработка событий.

1. Игра «Пятнашки» с произвольной размерностью.
2. Игра «Судоку» с произвольной размерностью.
3. Электронная таблица типа Excel с формулами – арифметические действия со значениями ячеек и константами.
4. Турнирная таблица - симметричное заполнение, определение суммы очков и места.
5. Игровой автомат с прокруткой картинок.
6. Редактор разметки для кнопок: создание, установка начального текста и иконки, перетаскивание по форме, изменение размеров.
7. Игра «Угадай букву» для произвольного слова.
8. Игра «Домино» - выставление костей в цепочку, кость – пара кнопок с иконками или изображение.
9. «Рулетка» - вращающееся колесо числами или иконками. При остановке выводится выпадающее число или иконка. При отсутствии средств поворота изображения перемещать горизонтально расположенное текстовое поле с числом или иконку по дуге окружности.
10. Аниматор сортировки: элементы сортировки – текстовые поля с числами. Графически отображается процесс перестановки элементов в массиве или перемещения из массива в массив

Лабораторная работа 2. Рефлексия. С использование средств рефлексии разработать универсальный сервис, применимый для объектов произвольного класса. Передается имя класса или его описатель класса Class.

1. Класс, выполняющий клонирование произвольного объекта, передаваемого по ссылке. Создает экземпляр объекта, просматривает список полей, копирует примитивные поля, для ссылочных типов – рекурсивно вызывает клонирование.
2. Редактор содержимого произвольного объекта, передаваемого по ссылке. Оконный класс со списком полей данных объекта - примитивных типов.
3. Класс - упорядоченный вектор по произвольному элементу данных класса - примитивного типа или строки, заданного по имени. Класс содержит вектор, ссылку на класс хранимых данных и имя поля, по которому производится

сортировка. Разработать метод сортировки, алгоритмы вставки с сохранением порядка и двоичного поиска.

4. Собственный JSON-сериализатор с сериализацией ссылок и массивов ссылок.
5. Собственный XML-сериализатор с сериализацией ссылок и массивов ссылок.
6. Класс – документатор. При вводе имени класса создает документ в формате pdf или html с описанием структуры объекта, а также тех классов, на которые в объекте имеются ссылки или массивы ссылок. Обработка производится рекурсивно, с исключением повторной обработки уже просмотренных классов.
7. Класс – документатор, аналогичный вар. 6, но для содержимого объекта. Вывод содержимого объекта и объектов, доступных по ссылкам и через массивы ссылок. Создает документ в формате pdf или html.
8. Класс – конечный автомат с вызовом методов, *прикрепляемых* к переходам и состояниям по имени. В описании автомата перечисляется список дуг в виде *входное состояние – входной символ – состояние перехода – имя метода, вызываемое при переходе*. Сами методы реализуются в производном классе для класса-автомата.
9. Собственный оконный класс с вызовом методов обработки событий по имени. Оконный класс, производный от JFrame, создает и размещает кнопки по задаваемым координатам и размерам, запоминает имя назначенного метода обработки клика. Перехватывает клик мыши по всей поверхности окна, идентифицирует кнопку по координатам и вызывает метод обработки по назначенному имени. Имена обработки и код инициализации кнопок пишется в производном классе.

Лабораторная работа 3. Разработка элемента GUI с динамическим поведением. Производный класс на основе базового элемента графического интерфейса (JPanel) с динамическим поведением. Класс способен накапливать, отображать и возвращать накопленные данные. При необходимости периодической передачи данных с определенной частотой получает при конструировании интерфейс слушателя события с методом *очередное значение данных*.

1. Полосковый индикатор уровня с установкой минимального и максимального порога и инертным индикатором максимума: текущий уровень – синяя полоска, максимальный – красная, линейно снижающаяся со времени. В момент превышения максимального порога генерирует событие.
2. Стрелочный инерционный прибор. При изменении значения имитирует и отображает затухающие колебания вокруг нового значения. При входе и выходе из красной зоны, обозначенной на индикаторе, генерирует событие.
3. График записываемых значений с отображением N последних записанных точек и *прокруткой* графика при выводе. Можно запросить время записи и значение N последних записанных точек.
4. Стрелочно-цифровой секундомер или часы с прогресс-индикатором отметки событий на циферблате. Событие содержит тестовое сообщение. Можно запросить любое из N последних событий.
5. Гистограмма, запоминающая количество отсчетов, их сумму, сумму квадратов и количество попаданий в карманы. Отображение текущего вида гистограммы. Можно запросить количество отсчетов в каждом кармане, среднее и дисперсию множества записанных значений.

6. Пиковый индикатор. При плавном изменении значений входного потока данных (амплитуды) зажигает на определенное время желтую лампочку и генерирует событие. При резком изменении – аналогичные действия для красной лампочки.
7. Светофор. Отображает светофор с состояниями *красный – желтый – зеленый – зеленый мигающий – желтый.* Программируется продолжительность фаз. При смене состояния генерирует событие. Цифровая индикация оставшегося времени для текущей фазы. Включение и выключение светофора. В выключенном состоянии – *желтый мигающий.*
8. Класс, производный от JFrame - jкно с плавающими элементами управления. При добавлении кнопок и текстовых полей, запоминает текущие размеры и координаты, содержит поток, который меняет координаты, моделируя прямолинейное движение с отражением от границ окна. Методы приостановки и возобновления движения, изменение скорости движения для отдельных объектов и одновременно для всех.

Лабораторная работа 4. Контроллер бизнес-логики с подпиской на события. Класс-контроллер реализует в отдельном потоке в реальном времени *модель поведения физического объекта.* Классы GUI (формы) получают ссылку на объект-контроллер с *интерфейсом команд* и *подписываются* на получение от него событий через соответствующий интерфейс (шаблон *наблюдатель*). При появлении в контроллере события последний вызывает соответствующий метод во всех подписанных объектах. При закрытии объекта класса GUI он *отписывается* от контроллера. События и команды в интерфейсах должны обеспечивать объектам внешнего представления полный набор возможностей управления и отображения состояния физического объекта.

1. Физический объект – кастрюля на плите: текущее положение конфорки, температура и объем содержимого, состояние нагревания и кипения, теплоотдача при открытой и закрытой крышке.
2. Физический объект – звуковая волна в формате wave-файла. Ручное проматывание и автоматическое проигрывание. Объекты – адаптеры для изменения амплитуды и задержки сигнала, микширования – суммирования сигналов от разных источников.
3. Физический объект – автомобиль, движущийся по прямой трассе со светофорами. Модели движения точечной массы с заданным положительным или отрицательным ускорением (педали *газа и тормоза*). Модель получает линейное расположение светофоров, возможно добавление светофора на указанном расстояния от последнего. Событие – *проезд на красный*, данные модели – текущая координата, скорость, расстояние до ближайшего светофора.
4. Физический объект – точка, движущаяся в гравитационном поле других стационарных точек в двумерной системе координат. Управление – ускорение и торможение по вектору скорости, поворот вправо/влево по вектору скорости на указанный угол.
5. Модель системы потребителей электроэнергии. Элемент – чайник с автоматическим выключением при закипании, лампочка, компьютер с источником бесперебойного питания. Потребители имеют устанавливаемую мощность и состояние включено/выключено, а также модель поведения в зависимости от наличия и отсутствия напряжение (нагрев и остывание чайника, работа бесперебойника при отключенном питании). Модели поведения потребителей также моделируются соответствующими контроллерами с внутренними потоками. Возможно подключение и отключение произвольного количества потребителей

разных типов. Фиксируется общее потребление и моделируется отключение сети при перегрузке.

6. Физическая модель взлетающего самолета. Текущая мощность двигателей расходуется на увеличение потенциальной энергии подъема и кинетической энергии. Подъемная сила за счет геометрии крыла и угла атаки, сила сопротивления воздуха пропорциональная скорости. Управление мощностью двигателя и углом атаки.

Лабораторная работа 5. Разработка многооконного интерфейса для л.р.3. Разработать многооконный GUI, в котором в каждом окне имеется оригинальный набор датчиков и элементов управления. Окна открываются из основного окна. Количество экземпляров окон не ограничено. Виды окон указаны для соответствующих вариантов предыдущей работы.

1. Общий вид с моделированием закипания в виде пузырьков пара, движок конфорки, цифровой индикатор температуры.
2. Окна для класса-источника сигнала и для классов-фильтров с отображением последних N значений сигнала на выходе.
3. Общий вид, вид из кабины водителя – размер и положение светофора в зависимости от расстояния с учетом перспективы, движки газа и тормоза.
4. Общий вид системы движущейся и стационарных точек, вид *из движущейся точки по направлению вектора скорости*, размер стационарных точек в зависимости от расстояния (перспектива), движки ускорения/торможения и поворота.
5. Окна для всех типов потребителей, окно силового щитка.
6. Общий вид, приборная панель самолета со всеми индикаторами и движками.

Лабораторная работа 6. Java Native Interface (JNI). Реализовать часть наиболее трудоемких методов в виде native-функций на Си++, сравнить производительность.

Лабораторная работа 7а. Потоковое программирование в приложении. Реализовать времяёмкие операции в виде отдельных потоков, параллельных потоку GUI, предусмотреть синхронизацию окончания операции, аварийное завершение, возможность периодического вывода параметров фонового потока в индикатор прогресса.

Лабораторная работа 7б. Тестовые, двоичные и сериализуемые потоки данных. Реализовать средства сохранения и загрузки полного состояния всех объектов программы в тестовый, двоичный и сериализуемый потоки.

Лабораторная работа 7в. Использование JDBC-библиотек для работы с базами данных. Реализовать средства сохранения и загрузки параметров настройки, собираемой статистики и прочих данных в БД SQLite.

Индивидуальные задания

Индивидуальное задание выполняется в виде разработки одного из шаблонов проектирования. Пояснительная записка должна содержать:

- учебно-методический материал по шаблону - учебники, учебные пособия, лекционный материал, тематические сайты;
- материал по практике применения шаблона - форумы, тематические сайты;
- описание разработки;
- результаты тестирования.

Все сторонние материалы должны быть снабжены корректно оформленными библиографическими ссылками.

Варианты заданий

1. *Шаблон MVC*. **Сетевая (локальная) игра типа «Морской бой», «домино».** Модель хранит структуру данных игры - расстановку кораблей, и над ней выполняются методы по управлению игрой. Контроллер определяет порядок ходов, проверяет возможность выполнения хода. Внешнее представление (view) связано с контроллером двунаправленным интерфейсом: контроллер управляет отображением элементов игрового поля, выводит текстовые сообщения, получает от представления события – клик по элементу поля, начало новой игры, завершение и т.п.. Варианты игры:
   - Локальная: одна модель, один контроллер, два представления для игроков
   - Две модели, два контроллера, два представления. Контроллеры поддерживают соединение и передают команды: сделан ход, синхронизация моделей, сброс и начальная установка игры
2. *Шаблон MVC (классический)*. Представление имеет ряд текстовых полей, отображающих параметры модели физического объекта. На каждый параметр представление отдельно подписывается к модели на событие, связанное с его изменением. Интерфейс подписки идентичен для всех параметров. Для каждого параметра создается оригинальный контроллер, обрабатывающий команду изменения значения параметра со стороны представления. Модель описывается набором зависимостей, параметры которой могут быть *входными, выходными (результатами) и выходными с возможностью задания начальных значений (инициализацией).*
3. *Шаблон – прототип.* **Класс таблицы с произвольной структурой столбцов**. Хранимые данные разных типов - целые, вещественные, дата, время, GPS-координаты создаются на основе абстракции данных с функционалом: имя класса, парсинг из сроки и вывод в строку, клонирование, сравнение и сложение с объектом того же типа. Строка таблицы определяется набором объектов-прототипов для столбцов. Сама строка также клонируется. Таблица состоит из вектора строк-имен, строки-прототипа и строк самой таблицы. Функционал: создание таблицы, добавление столбцов, добавление строк, сортировка по столбцу с заданным номером, сложение строк, сохранение и загрузка из файла. Программный и интерфейс и оконное приложение. Тестирование на больших данных - импорт из Excel, генерация тестовых таблиц.
4. *Шаблон – приспособленец*. **Работа с деревом версий текстового файла**. Файл редактируется по словам. Класс словаря содержит хэш-таблицу адаптеров со ссылками на оригинальные слова в виде *ключ – само слово*. Адаптер содержит количество ссылок на слово из всех версий текста. Каждая версия текста – вектор ссылок на адаптеры. При добавлении или вставке слова в версию текста оно ищется в фабрике, при нахождении счетчик в адаптере увеличивается на 1. Иначе добавляется в словарь с новым адаптером. При удалении счетчик ссылок уменьшается. Изменение слова рассматривается как последовательность операций удаления и вставки. Вся структура данных сериализуется в файл. Протестировать шаблон на сказке *Репка*.
5. *Шаблон – композиция для древовидной системы.* **Графический редактор** с группировкой/разгруппировкой элементов, изменением размеров и порядка отображения - перенос на передний/задний план, перетаскиванием объектов, сохранением картинки в файл. Абстрактный класс элемента, группы элементов и

конкретных графических объектов - окружность, полигон, строка текста. Ограничивающий прямоугольник, селекция выбора объектов по точке и прямоугольнику.

6. *Шаблон Command*. Группа команд обработки объекта с общим интерфейсом *Do/ReDo/UnDo*. Производный класс запоминает параметры, необходимые для выполнения прямой и обратной команды. Класс – менеджер команд поддерживает очередь команд ограниченной длины, текущий обрабатываемый объект, методы *Do/ReDo/UnDo*, выбирая их из очереди и вызывая соответствующие методы в объектах-командах. **Графический редактор** набором команд редактирования графических объектов - создать объект, переместить, изменить размер, удалить, переместить на передний и задний план.
7. *Шаблон – пул потоков*. Класс потока представляет собой поток, запускаемый при создании объекта. Содержит ссылку на исполняемый код через *Runnable*, код завершения и ссылку на менеджер пула. Код потока содержит цикл, в котором засыпает, после пробуждения выполняет исполняемый код и уведомляет менеджер о своем завершении и засыпает. Менеджер потоков содержит вектор таких потоков. При обращении к менеджеру с запросом, содержащим код исполнения и код завершения, выбирается поток из пула, заполняется данными и пробуждается. Если свободного потока нет, то запрос ставится в очередь. Провести сравнительное тестирование для потока запросов с пулом и при запуске потоков обычным образом.
8. *Шаблон – прокси (фильтр)*. Класс с интерфейсом текстового потока при конструировании делегируется к однотипному объекту источнику и **отфильтровывает набор слов,** передаваемый при конструировании, используется внутренняя очередь символов для отложенного распознавания. Протестировать на цепочке фильтров для разных наборов слов.
9. *Шаблон – сессия* для соединения на сокетах в виде библиотеки для клиента/сервера. Клиент и сервер используют синхронный обмен *запрос-ответ*. При первоначальном установлении соединения клиент получает уникальный идентификатор, сервер создает дескриптор соединения. Клиент нумерует передаваемые сообщения, сервер сохраняет номер и ответ на последнее переданное сообщение. Сервер периодически или после каждого изменения сохраняет дескрипторы в файл. Клиент также запоминает в файле идентификатор сессии, номер и само последнее переданное сообщение. Обеспечить сохранность последовательности (отсутствие пропадания и дублирования) сообщений при перезагрузке клиента и сервера. Реализовать модель **банкомата и платежной системы.**
10. *Модель параллельных запросов к серверу от группы потоков.* Множество потоков может посылать независимые запросы к серверу через единственное соединение. Запрос ставится в очередь, нумеруется, клиентский процесс засыпает. Поток передачи от клиента выбирает сообщения из очереди и передает в соединение. Поток приема на сервере, приняв очередное сообщение, запускает поток исполнения запроса, по завершении которого в очередь ответов ставится ответ, в котором сохраняется порядковый номер запроса. Поток передачи ответов на сервере выбирает сообщения из очереди и передает в соединение. Поток приема ответов определяет по номеру в сообщении поток, передавший запрос и пробуждает его. Ответное сообщение находится в запросе, переданном потоком, и выводится им. Промоделировать **группу потоков, посылающих случайные слова, которые сервер переворачивает со случайной задержкой**.

11. *Шаблон – кэш объектов*. Разработать класс - кэш объектов с возможностью изменения размера кэша, сбора статистики и применения различных стратегий вытеснения (FIFO, LRU, RAND). Использовать для кэширования слов текстового файла при последовательном чтении. Кэш – хеш-таблица с ключом-словом и необходимыми параметрами для моделирования, например, номер последнего обращения для LRU. При чтении очередного слова проверяется его наличие в кэше. При отсутствии производится замещение. Для выбранного файла строится зависимость доли попадания в кэш в зависимости от размера кэша и способа вытеснения. Сравнить результаты для файлов с разным содержимым - художественное произведение, технический текст.

12. *Шаблон – итератор*. Операции над итератором: установка на первый, последний, следующий, предыдущий, по логическому номеру, извлечение через итератор, вставка на позицию итератора, замещение, удаление. Структура данных: двухуровневый массив ссылок (двумерный массив), двоичное дерево, дерево с данными в конечных вершинах, список с массивом ссылок [3-2]. Использование нескольких итераторов. ***Замечание по теме**:* для корректной реализации операции удаления в основном классе использовать вектор созданных итераторов. При удалении одним из них элемента остальные, которые на него ссылаются, генерируют исключение при очередном обращении. Рассмотреть другие варианты корректного разделения.

Разработка классов, использующих внутренние потоки для промежуточной буферизации данных. Для моделирования прикладных процессов использовать потоки, которые засыпают на случайный момент времени, после чего читают/записывают очередную порцию данных постоянного или случайного размера. Предусмотреть сбор статистики в классах буферизации – средний объем данных в буфере.

13. Класс буферизованного ввода в реальном времени. При конструировании получает параметр – физический поток данных с интерфейсом *InputStream*. Использует внутренний циклический буфер или односвязный список блоков, содержащих массив байтов фиксированной размерности (буферный пул). Создает поток, который читает байты из входного потока и записывает в циклический буфер. При заполнении циклического буфера засыпает. Метод чтения извлекает из циклического буфера очередной байт, возвращает -1 при окончании данных в потоке-источнике, блокируется при отсутствии данных в циклическом буфере.

14. Класс – *PipedStream* с циклическим буфером данных. Класс *PipedOutputStream* имеет циклический буфер, в который пишет поток байтов, блокируя текущий поток при заполнении буфера. Класс *PipeInputStream* получает ссылку на *PipedOutputStream* и при чтении данных либо блокируется при их отсутствии, либо извлекает данные, деблокируя поток записи.

15. Класс отложенной записи. При открытии файла классом с присоединенным интерфейсом *OutputStream* создается *ByteOutputStream*, в который пишутся данные потока. При закрытии объект, содержащий имя файл и байтный массив ставится в очередь, из которой фоновый поток их извлекает и пишет файлы. Сравнить среднее время записи в обычный и *отложенный файл.*