Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Новосибирский государственный технический университет»
Кафедра вычислительной техники

Паспорт
расчетно-графического задания (работы)

по дисциплине «Технология разработки программного обеспечения», 1 семестр

1. Методика оценки

Расчетно-графическое задание выполняется в виде разработки одного из шаблонов проектирования. Пояснительная записка должна содержать:

- Учебно-методический материал по шаблону (учебники, учебные пособия, лекционный материал – краткое описание)
- Материал по практике применения шаблона (форумы, тематические сайты)
- Описание разработки
- Результаты тестирования
  Все сторонние материалы должны быть снабжены корректно оформленными библиографическими ссылками.

При защите расчетно-графического задания (работы) оцениваются следующие виды работ:

- Постановка и анализ задачи;
- Разработка структур данных для представления предметной области и алгоритмов;
- Объем и качество программного кода;
- Тестирование программного кода;
- Оформление текста программы и результатов работы

2. Критерии оценки

Согласно положению о балльно-рейтинговой системе НГТУ, базовый балл рейтинга за РГР определен в рабочей программе (16 баллов). Соответственно, критерий оценки определяется в процентах к этому баллу:

- РГР считается **невыполненной,** если отсутствует необходимые разделы описания программы, либо программа не работает, либо студент не в состоянии объяснить принципов ее работы (алгоритма) – оценка составляет <25% базовой;
- РГР засчитывается на **пороговом** уровне, разработка не полностью соответствует заданию, имеются ошибки при тестировании, пояснительная записка оформлена со значительными структурными, стилистическими и грамматическими ошибками - оценка составляет не более 50% базовой
- РГР засчитывается на **базовом** уровне, если функционал разработки соответствует заданию, отсутствуют ошибки тестирования, пояснительная записка оформлена в целом грамотно - оценка составляет 50-80% базовой
- РГР засчитывается на **продвинутом** уровне, если функционал реализован полностью и эффективно, в пояснительной записке отражены все аспекты

структурного, функционального проектирования и тестирования - оценка составляет 80-100% базовой

3. **Шкала оценки**

В общей оценке по дисциплине баллы за РГЗ(Р) учитываются в соответствии с правилами балльно-рейтинговой системы, приведенными в рабочей программе дисциплины (16 баллов).

4. **Примерный перечень тем РГЗ(Р)**

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;
   - сетевая: одна модель, один контроллер, два представления для игроков. Представления связываются с приложением, содержащим контроллер и модель. Контроллер обеспечивает множественные соединения и синхронизацию;
   - сетевая: 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 строки 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, переносом на передний/задний план, перемещением объектов, сохранением картинки в файл. Абстрактный класс элемента, группы элементов и конкретных графических объектов - окружность, полигон, строка текста. Ограничивающий прямоугольник, селекция 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 исполняемый назначенный код, уведомляет менеджер о своем завершении и засыпает. Менеджер потоков содержит вектор потоков-исполнителей. 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 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 как функция 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 [9]. 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.*

2.