Практическая работа №11

Задания на очереди

Теоретические сведения (что нужно знать)

1 Классы
- ✓ **Инвариант класса**
- ✓ **Задачи инкапсуляции**

2 Интерфейсы
- ✓ **Интерфейс как синтаксический контракт**
- ✓ **Интерфейс как семантический контракт**

3 Абстрактные базовые классы и наследование
- ✓ **Устранение дублирования**
- ✓ **Вынос изменяемой логики в наследников**

Задания

Задание 1. Очередь на массиве

- **Найдите инвариант структуры данных «очередь». Определите функции, которые необходимы для реализации очереди. Найдите их пред- и постусловия.**
- **Реализуйте классы, представляющие циклическую очередь с применением массива.**
  - **Класс `ArrayQueueModule` должен реализовывать один экземпляр очереди с использованием переменных класса.**
  - **Класс `ArrayQueueADT` должен реализовывать очередь в виде абстрактного типа данных (с явной передачей ссылки на экземпляр очереди).**
  - **Класс `ArrayQueue` должен реализовывать очередь в виде класса (с неявной передачей ссылки на экземпляр очереди).**
  - **Должны быть реализованы следующие функции(процедуры) / методы:**
    - **`enqueue` – добавить элемент в очередь;**
    - **`element` – первый элемент в очереди;**
    - **`dequeue` – удалить и вернуть первый элемент в очереди;**
    - **`size` – текущий размер очереди;**
    - **`isEmpty` – является ли очередь пустой;**
    - **`clear` – удалить все элементы из очереди.**
  - **Инвариант, пред- и постусловия записываются в исходном коде в виде комментариев.**
  - **Обратите внимание на инкапсуляцию данных и кода во всех трех реализациях.**
- **Напишите тесты реализованным классам.**

Задание 2. Очереди

4 Определите интерфейс очереди `Queue` и опишите его контракт.

5 Реализуйте класс `LinkedQueue` — очередь на связном списке.

6 Выделите общие части классов `LinkedQueue` и `ArrayQueue` в базовый класс `AbstractQueue`.

Дополнительные задания

Задание 3. Вычисление выражений

1. **Разработайте классы `Const`, `Variable`, `Add`, `Subtract`, `Multiply`, `Divide` для вычисления выражений с одной переменной.**
2. **Классы должны позволять составлять выражения вида**

```
new Subtract(
            new Multiply(
                new Const(2),
                new Variable("x")
                ),
                 new Const(3)
           ).evaluate(5)
```

При вычислении такого выражения вместо каждой переменной подставляется значение, переданное в качестве параметра методу `evaluate` (на данном этапе имена переменных игнорируются). Таким образом, результатом вычисления приведенного примера должно стать число 7.

3. **Для тестирования программы должен быть создан класс `Main`, который вычисляет значение выражения x^2-2x+1, для `x`, заданного в командной строке.**
4. **При выполнение задания следует обратить внимание на:**
    - **Выделение общего интерфейса создаваемых классов.**
    - **Выделение абстрактного базового класса для бинарных операций.**

Задание 4

1. **Доработайте предыдущее задание, так что бы выражение строилось по записи вида**

```
x * (y - 2)*z + 1
```

Для этого реализуйте класс `ExpressionParser` с методом `TripleExpression parse(String)`.

2. **В записи выражения могут встречаться: умножение `*`, деление `/`, сложение `+`, вычитание `-`, унарный минус `-`, целочисленные константы (в десятичной системе счисления, которые помещаются в 32-битный знаковый целочисленный тип), круглые скобки, переменные (`x`, `y` и `z`) и произвольное число пробельных символов в любом месте (но не внутри констант).**
3. **Приоритет операторов, начиная с наивысшего**
    - **унарный минус;**
    - **умножение и деление;**
    - **сложение и вычитание.**
4. **Для выражения `1000000*x*x*x*x*x/(x-1)` вывод программы должен иметь следующий вид:**

```
x       f
     0       0
     1       division by zero
     2       32000000
```

```
3       121500000
4       341333333
5       overflow
6       overflow
7       overflow
8       overflow
9       overflow
10      overflow
```

5. **Результат `division by zero` (`overflow`) означает, что в процессе вычисления произошло деление на ноль (переполнение).**
6. **Разбор выражений рекомендуется производить методом рекурсивного спуска. Алгоритм должен работать за линейное время.**

7. **При выполнении задания следует обратить внимание на дизайн и обработку исключений.**

8. **Человеко-читаемые сообщения об ошибках должны выводится на консоль.**
9. **Программа не должна «вылетать» с исключениями (как стандартными, так и добавленными).**

ПРАКТИЧЕСКАЯ РАБОТА №12 ОБРАБОТКА СТРОК.

ЦЕЛЬ ПРАКТИЧЕСКОЙ РАБОТЫ:

Цель данной лабораторной работы – закрепить знания в области обработки строк, научиться применять методы класса String и других классов для обработки строк.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Для работы с текстовыми данными в Java есть три класса: String, StringBuffer и StringBuilder.

Особенности использования строк в Java

В Java строки представляют собой неизменяемую последовательность символов Unicode. В отличие от представления в C / C ++, где строка является просто массивом типа char, любая Java, строка является объектом класса java.lang.

Однако Java строка, представляет собой в отличие от других используемых классов особый класс, который обладает довольно специфичными характеристиками. Отличия класса строк от обычных классов:

- строка в Java представляет из себя строку литералов (текст), помещенных в двойные кавычки, например:

"Hello , World! ". Вы можете присвоить последовательность строковых литералов непосредственно переменной типа String, вместо того чтобы вызывать конструктор для создания экземпляра класса String.

- Оператор '+' является перегруженным, для объектов типа String, и всегда используется, чтобы объединить две строки операндов. В данном контексте мы говорим об операции конкатенации или сложения строк. Хотя '+' не работает как оператор сложения для любых других объектов, кроме строк, например, таких как Point и Circle.
- Строка является неизменяемой, то есть, символьной константой. Это значит, что ее содержание не может быть изменено после ее (строки как объекта) создания. Например, метод toUpperCase () – преобразования к верхнему регистру создает и возвращает новую строку вместо изменения содержания существующей строки.

Обратитесь к API JDK для того чтобы ознакомиться с полным списком возможностей класса String в java.lang.String. Наиболее часто используемые методы класса String приведены ниже.

```
//длина
int length()       // возвращает длину String
boolean isEmpty()  // то же самое thisString.length == 0
// сравнение
boolean equals(String another)
// НЕЛЬЗЯ использовать'==' или'!=' для сравнения объектов String в Java
boolean equalsIgnoreCase(String another)
int compareTo(String another)// возвращает 0 если эта строка совпадает с another;
// <0 если лексикографически меньше another; or >0
int compareToIgnoreCase(String another)
boolean startsWith(String another)
boolean startsWith(String another, int fromIndex)  // поиск начинается с fromIndex

boolean endsWith(String another)

// поиск & индексирование
int indexOf(String search)
int indexOf(String search, int fromIndex)
int indexOf(int character)
```

```
int indexOf(int character, int fromIndex)      // поиск вперед от fromIndex
int lastIndexOf(String search)
int lastIndexOf(String search, int fromIndex)  // поиск назад от fromIndex
int lastIndexOf(int character)
int lastIndexOf(int character, int fromIndex)

// выделение char или части строки из String (подстрока)
char charAt(int index)            // позиция от 0 до  (длина строки-1)
String substring(int fromIndex)
String substring(int fromIndex, int endIndex)  // exclude endIndex

// создается новый String или char[] из исходного (Strings не изменяются!)
String toLowerCase() //преобразование к нижнему регистру
String toUpperCase() //преобразование к верхнему регистру
String trim()       // создается новый String с помощью удаления пробелов спереди и сзади
String replace(char oldChar, char newChar)  // создание нового String со старым  oldChar перемещается посредством буфера newChar
String concat(String another)             // то же самое как thisString + другое
char[] toCharArray()                    // создается char[] из string
void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)  // копируется в массив назначения dst char[]

// статические методы для преобразования примитивов в String
static String ValueOf(type arg)  // тип может быть примитивный или char[]

// статические методы дают форматированный String используя //спецификаторы форматирования
static String format(String formattingString, Object... args)// так же как printf()

// регулярные выражения (JDK 1.4)

boolean matches(String regexe)

String replaceAll(String regexe, String replacement)

String replaceAll(String regexe, String replacement)

String[] split(String regexe) // разделяет String используя regexe как разделитель,

// возвращает массив String

String[] split(String regexe, int count)  // для подсчета количества раз только (count)
```

Статический метод String.format()

Статический метод String.format() (введен в JDK 1.5) может быть использован для получения форматированного вывода, таким же образом как это делается в языке Си с использованием функции printf() и спецификаторов вывода для различных типов данных. Метод format() делает то же самое, что функция printf(). Например**:**

```
String.format("%.1f", 1.234);   // возвращает String "1.2"
```

Удобно использовать String.format(), если вам нужно получить простую отформатированную строку для некоторых целей (например, для использования в методе ToString()). Для сложных строк, нужно использовать StringBuffer / StringBuilder с Formatter . Если вам просто нужно отправить простую отформатированную строку на консоль, то просто воспользуйтесь методом System.out.printf(), например:

```
System.out.printf("%.1f", 1.234);
```

Пример использования методов lastIndexOf() и substring() в пользовательском классе Filename

```
public class Filename {
   private String fullPath;
   private char pathSeparator, extensionSeparator;
   public Filename(String str, char sep, char ext) {
      fullPath = str;
      pathSeparator = sep;
      extensionSeparator = ext;
   }
   public String extension() {
      int dot = fullPath.lastIndexOf(extensionSeparator);
return fullPath.substring(dot + 1);
   }
    // получение имени файла без расширения
   public String filename() {
      int dot = fullPath.lastIndexOf(extensionSeparator);
      int sep = fullPath.lastIndexOf(pathSeparator);
      return fullPath.substring(sep + 1, dot);
   }
   public String path() {
      int sep = fullPath.lastIndexOf(pathSeparator);
      return fullPath.substring(0, sep);
   }
```

Теперь рассмотрим программу, которая использует класс Filename:

```
public class FilenameTester {
   public static void main(String[] args) {
      final String FPATH = "/home/user/index.html";
      Filename myHomePage = new Filename(FPATH, '/', '.');
      System.out.println("Extension = " + myHomePage.extension());
```

```
        System.out.println("Filename = " + myHomePage.filename());

        System.out.println("Path = " + myHomePage.path());

    }

}
```

Результат работы программы:
Extension = html
Filename = index
Path = /home/user

Особенности класса String

Строки получили специальное значение в Java, потому что они часто используются в любой программе. Следовательно, эффективность работы с ними (с точки зрения вычислений и хранения) имеет решающее значение. Что нужно помнить про класс **String**

- это immutable (неизменный) класс
- это final класс

Разработчики Java все-таки решили сохранить примитивные типы в объектно-ориентированном языке вместо того, чтобы сделать вообще все в виде объектов. Нужно сказать, что сделано это в первую очередь, для того чтобы повысить производительность языка. Ведь примитивы хранятся в стеке вызовов, и следовательно, требуют меньше пространства для хранения, и ими легче управлять. С другой стороны, объекты хранятся в области памяти, которую используют программы, и которая называется “куча” (heap), а этот механизм требуют сложного управления памятью и потребляет гораздо больше места для хранения.

По соображениям производительности, класс String в Java разработан, так, чтобы быть чем-то промежуточным между примитивными типами данных и типами данных типа класс. Как уже было отмечено выше специальные характеристики типа String включают в себя:

- '+' оператор, который выполняет сложение примитивных типов данных (таких, как int и double), и перегружен, чтобы работать на объектах String. Операция '+’ выполняет конкатенацию двух операндов типа String.
- Java не поддерживает механизма перегрузки операций по разработке программного обеспечения. В языке, который поддерживает перегрузку операций, например C ++, вы можете превратить оператор '+' (с помощью перегрузки) в оператор для выполнения сложения или вообще вычитания, например двух матриц, кстати это будет примером плохого кода. В Java оператор '+' является единственным оператором, который внутренне перегружен, чтобы поддержать конкатенацию (сложение) строк в Java. Нужно принять к сведению, что '+' не работает на любых других произвольных объектах, помимо строк, например, таких как рассмотренные нами ранее классы Point или Circle.

Теперь, собственно, о строках. Существуют несколько способов создания строк. Строка String может быть получена одним из способов:

- непосредственно из присвоения строкового литерала ссылке типа String – таким же способом как примитивные типы данных;
- или с помощью оператора new и конструктора класса String, аналогично вызову конструктора любого другого класса. Тем не менее, этот способ не часто используется и использовать его не рекомендуется.

Для примера:

```
String str1 = "Java is Hot";          // неявный вызов конструктора через присваивание строкового литерала

String str2 = new String("I'm cool");  // явный вызов конструктора через new

char[] array = { 'h', 'e', 'l', 'l', 'o', '.' }; //еще один способ создания строки
```

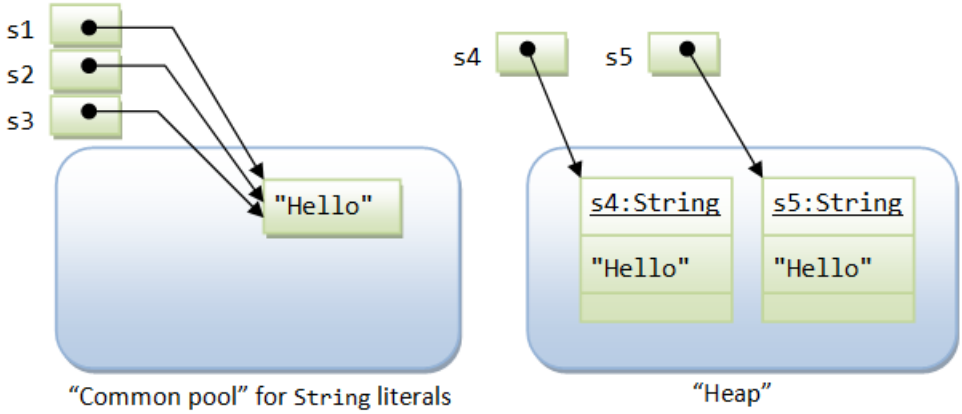
В первом случае str1 объявлена как ссылка типа String и инициализируется строкой “Java is Hot”. Во второй строке, str2 объявлена как ссылка на строку и инициализируется с помощью вызова оператора new и конструктора, который инициализирует ее значением “I'm cool”. Строковые литералы хранятся в общем пуле. Это облегчает совместное использование памяти для строк с тем же содержанием в целях сохранения памяти. Объекты строк, выделенные с помощью оператора new, хранятся в куче (heap), а там нет разделяемого хранилища для того же самого контента (содержания).

Строковые литералы и объекты типа String

Как уже упоминалось, есть два способа создания строк: неявное создание путем присвоения строкового литерала переменной или явного создания объекта String, через вызов оператора new и вызов конструктора. У класса String есть еще одна особенность. Все строковые литералы, определенные в Java коде, вроде "asdf", на этапе компиляции кэшируются и добавляются в так называемый пул строк. Например:

```
String s1 = "Hello";              // String литерал

String s2 = "Hello";              // String литерал

String s3 = s1;                   // одинаковые ссылки

String s4 = new String("Hello");  // String объект

String s5 = new String("Hello");  // String объект
```

Java предоставляет специальный механизм для хранения последовательностей символьных литералов (строк), так называемый общий пул строк. Если две последовательности литералов (строки) имеют одинаковое содержание, то они разделяют общее пространство для хранения внутри общего пула. Такой подход принят для того, чтобы сохранить место для хранения часто используемых строк. С другой стороны, объекты типа String (строки), созданные с помощью оператора new и конструктора хранятся в куче.



Каждый объект String в куче имеет свое собственное место для хранения, как и любой другой объект. Там нет обмена хранения в куче, даже если два объекта Строковые имеют то же содержание.

А в куче нет разделяемого пространства для хранения двух объектов, даже если эти два объекта являются объектами типа String и имеют одинаковое содержание.

Вы можете использовать метод equals() класса String для сравнения содержимого двух строк. Вы можете использовать оператор сравнения на равенство '==', чтобы сравнить ссылки (или указатели) двух объектов. Изучите следующие программные примеры:

```
s1 == s1;        // true, одинаковые ссылки

s1 == s2;        // true, s1 and s1 разделяют общий пул

s1 == s3;        // true, s3 получает то же самое значение что ссылка s1

s1.equals(s3);   // true, одинаковое содержимое

s1 == s4;        // false, различные ссылки

s1.equals(s4);   // true, одинаковое содержимое

s4 == s5;        // false, различные ссылки в куче

s4.equals(s5);   // true, одинаковое содержимое
```

Важные замечания

- В приведенном выше примере, используется оператор отношения для того чтобы проверить на равенство '==' ссылки двух объектов String. Это сделано, чтобы показать различия между строковыми последовательностями литералов, которые используют совместное пространство для хранения в общем пуле строк и объектов String, созданных в куче. Это логическая ошибка в использовании выражения (str1 == str2) в программе, чтобы сравнить содержимое двух объектов типа String.
- Строка может быть создана непосредственно путем присваивания последовательности литералов (строки), которая разделяет общий пул строк. Не рекомендуется использовать оператор new для создания объектов String в куче.

String является неизменяемыми

С тех самых пор, когда в языке Java появились возможности по использованию разделяемого пространства для хранения строк с одинаковым содержанием в виде строкового пула, String в Java стали неизменяемыми. То есть, как только строка создается (как объект в памяти программы), ее содержание не может быть изменено никаким образом (по аналогии с Си – строковые литералы — это символьные константы). В противном случае, если этого не сделать, другие ссылки String разделяющие ту же самую ячейку памяти будут зависеть от изменений, которые могут быть непредсказуемыми и, следовательно, является нежелательными. Такой метод, как например, toUpperCase () казалось-бы может изменить содержимое объекта String. Хотя на самом деле, создается совершенно новый объект String и возвращается как раз он в точку вызова. Исходный объект-строка будет впоследствии удален сборщиком мусора (Garbage-collected), как только не окажется больше ссылок, которые ссылаются на него.

Вот поэтому-то объект типа String и считается неизменяемым объектом, вследствие этого, считается не эффективным использовать тип String, например, в том случае, если вам нужно часто модифицировать строку (так вы в таком случае будете создавать много новых объектов типа String, которые каждый раз будут занимать новые места для хранения). Например,

```
// неэффективный код
String str = "Hello";
for (int i = 1; i < 1000; ++i) {
  str = str + i;
}
```

Замечание.

Если содержимое строки должно часто меняться в вашей программе, используйте классы StringBuffer или StringBuilder вместо класса String.

Классы StringBuffer и StringBuilder

Классы StringBuffer и StringBuilder в Java используются, когда возникает необходимость сделать много изменений в строке символов.В отличие от String, объекты типа StringBuffer и StringBuilder могут быть изменены снова и снова. В Java StringBuilder был введен начиная с Java 5.

Как объяснялось выше, строки String являются неизменяемыми, поэтому строковые литералы с таким контентом хранятся в пуле строк. Изменение содержимого одной строки непосредственно может вызвать нежелательные побочные эффекты и может повлиять на другие строки, использующие ту же память.

JDK предоставляет два класса для поддержки возможностей по изменению строк: это классы StringBuffer и StringBuilder (входят в основной пакет java.lang).

Объекты StringBuffer или StringBuilder так же, как и любые другие обычные объекты, которые хранятся в куче, а не совместно в общем пуле, и, следовательно, могут быть изменены, не вызывая нехороших побочных эффектов на другие объекты.

Класс StringBuilder как класс был введён в JDK 1.5. Это то же самое, как использование класса StringBuffer, за исключением того, что StringBuilder не синхронизирован по многопоточным операциям. Тем не менее, для программы в виде одного потока или нити управления, использование класса StringBuilder, без накладных расходов на синхронизацию, является более эффективным.

Использование StringBuffer

Класс находится в пакете **java.lang**. Прочитайте спецификацию API JDK для использования java.lang.StringBuffer.

Методы класса:

```
// конструкторы
StringBuffer()            // инициализация пустым anStringBuffer
StringBuffer(int size)    // определяет размер при инициализации
StringBuffer(String s)    //инициализируетя  содержимым   s
// Length
int length()

// Методы для конструирования содержимого
StringBuffer append(type arg)  // тип может быть примитивным, char[], String, StringBuffer, и т.д.
StringBuffer insert(int offset, arg)

// Методы для манипуляции содержимым
StringBuffer delete(int start, int end)
StringBuffer deleteCharAt(int index)
void setLength(int newSize)
void setCharAt(int index, char newChar)
StringBuffer replace(int start, int end, String s)
StringBuffer reverse()

// Методы для выделения целого/части содержимого
char charAt(int index)
String substring(int start)
String substring(int start, int end)
String toString()
// Методы для поиска
int indexOf(String searchKey)
int indexOf(String searchKey, int fromIndex)
int lastIndexOf(String searchKey)

int lastIndexOf(String searchKey, int fromIndex)
```

Обратите внимание, что объект класса StringBuffer является обычным объектом в прямом понимании этого слова. Вам нужно будет использовать конструктор для создания объектов типа класс StringBuffer (вместо назначения в строку буквальном). Кроме того, оператор '+' не применяется к объектам, в том числе и

к объектам StringBuffer. Вы должны будете использовать такой метод, как append() или insert() чтобы манипулировать StringBuffer.

Чтобы создать строку из частей, более эффективно использовать класс StringBuffer (для многопоточных программ) или StringBuilder (для однопоточных), вместо конкатенации строк. Например,

```
// Создадим строку типа YYYY-MM-DD HH:MM:SS

int year = 2010, month = 10, day = 10;

int hour = 10, minute = 10, second = 10;

String dateStr = new
StringBuilder().append(year).append("").append(month).append("").append(day).append("").append(hour)

.apend(":").append(minute).append(":").append(second).toString();

System.out.println(dateStr);


// StringBuilder более эффективная конкатенация строк

String anotherDataStr = year + "-" + month + "-" day + " " + hour + ":" + minute + ":" + second;

System.out.println(anotherDataStr);
```

Компилятор JDK, по сути, использует оба класса как String, так и StringBuffer для обработки конкатенации через операцию сложения строк '+'.

Для примера, рассмотрим строчку кода:

```
String msg = "a" + "b" + "c";
```

Представленная выше строчка кода будет скомпилирован в следующую для повышения эффективности:

```
String msg = new StringBuffer().append("a").append("b").append("c").toString();
```

В этом процессе создаются промежуточный объект StringBuffer и возвращаемый объект String.

Использование класса StringBuilder

Класс находится в пакете **java.lang**. Прочитайте спецификацию API JDK для использования. java.lang.StringBuilder

Программа ниже демонстрирует разные способы инвертирования длинных строк. Сравниваются три способа работы со строками: как с объектами класса String, так и с помощью StringBuffer и StringBuilder с использованием метода reverse(). Для измерения времени выполнения различных участков кода в примере используется метод

public static native long nanoTime();

Этот метод возвращает текущее значение наиболее точное время системных часов (таймера), в наносекундах. Его можно использовать только для измерения затраченного времени на выполнение операций, и он никак не связан с системным временем и текущим мировым временем.

Возвращаемое методом значение представлено в виде наносекунд, с момента фиксации, на любой произвольный момент времени. Например, нам необходимо определить, сколько времени занимает выполнение некоторого кода, для этого необходимо выполнить следующее:

```
/*

    *   long startTime = System.nanoTime();

    *   // ... the code being measured ...

    *   long estimatedTime = System.nanoTime() - startTime;
```

```
     *
     *
     * @return The current value of the system timer, in nanoseconds.
     * @since 1.5
     */
// Риверс длинной строки с помощью String и StringBuffer
public class StringsBenchMark {
   public static void main(String[] args) {
      long beginTime, elapsedTime;

      // Build a long string
      String str = "";
      int size = 16536;
      char ch = 'a';
      beginTime = System.nanoTime();   // Эталонное время в наносекундах
      for (int count = 0; count < size; ++count) {
        str += ch;
        ++ch;
        if (ch > 'z') {
          ch = 'a';
        }
      }
elapsedTime = System.nanoTime() - beginTime;
      System.out.println("Elapsed Time is " + elapsedTime/1000 + " usec (Build String)");

      // Риверс строки, строим другую строку за символом в обратном порядке
      String strReverse = "";
      beginTime = System.nanoTime();
      for (int pos = str.length() - 1; pos >= 0 ; pos--) {
        strReverse += str.charAt(pos);   // Конкатенация
      }
      elapsedTime = System.nanoTime() - beginTime;
      System.out.println("Elapsed Time is " + elapsedTime/1000 + " usec (Using String to reverse)");

      // Риверс строки через пустой StringBuffer путем добавления символов в обратном порядке
```

```
    beginTime = System.nanoTime();
    StringBuffer sBufferReverse = new StringBuffer(size);
    for (int pos = str.length() - 1; pos >= 0 ; pos--) {
      sBufferReverse.append(str.charAt(pos));      // append
    }
    elapsedTime = System.nanoTime() - beginTime;
System.out.println("Elapsed Time is " + elapsedTime/1000 + " usec (Using StringBuffer to reverse)");

    // Reverse a String by creating a StringBuffer with the given String and invoke its reverse()
    beginTime = System.nanoTime();
    StringBuffer sBufferReverseMethod = new StringBuffer(str);
    sBufferReverseMethod.reverse();     // use reverse() method
    elapsedTime = System.nanoTime() - beginTime;
    System.out.println("Elapsed Time is " + elapsedTime/1000 + " usec (Using StringBuffer's reverse() method)");

    // Reverse a String via an empty StringBuilder by appending characters in the reverse order
    beginTime = System.nanoTime();
    StringBuilder sBuilderReverse = new StringBuilder(size);
    for (int pos = str.length() - 1; pos >= 0 ; pos--) {
      sBuilderReverse.append(str.charAt(pos));
    }
    elapsedTime = System.nanoTime() - beginTime;
    System.out.println("Elapsed Time is " + elapsedTime/1000 + " usec (Using StringBuilder to reverse)");

    // Reverse a String by creating a StringBuilder with the given String and invoke its rever its reverse()
    beginTime = System.nanoTime();
    StringBuffer sBuilderReverseMethod = new StringBuffer(str);
    sBuilderReverseMethod.reverse();
    elapsedTime = System.nanoTime() - beginTime;
    System.out.println("Elapsed Time is " + elapsedTime/1000 + " usec (Using StringBuidler's reverse() method)");
  }
}
```

Задачи

Задание 1. (10%)

Разработать класс Person, в котором имеется функция, возвращающая Фамилию И.О. Функция должна учитывать возможность отсутствия значений в полях Имя и Отчество. Программу оптимизировать с точки зрения быстродействия.

Задание 2. (20%)

Доработать класс адреса, который из полученной строки формата "Страна[d] Регион[d] Город[d] Улица[d] Дом[d] Корпус[d] Квартира" ([d] – разделитель, например, «запятая») выбирает соответствующие части и записывает их в соответствующие поля класса Address. Учесть, что в начале и конце разобранной части адреса не должно быть пробелов. Все поля адреса строковые. Разработать проверочный класс не менее чем на четыре адресных строки. В программе предусмотреть две реализации этого метода:
а) разделитель – только запятая (использовать метод split()); Внимание, при разработке нужно учесть, что
б) разделитель – любой из символов „.;- (класс StringTokenizer).

Задание 3. (30%)

Реализуйте класс Shirt: Метод toString() выводит объяснение и значение полей построчно.

Дан также строковый массив: shirts[0] = "S001,Black Polo Shirt,Black,XL"; shirts[1] = "S002,Black Polo Shirt,Black,L"; shirts[2] = "S003,Blue Polo Shirt,Blue,XL"; shirts[3] = "S004,Blue Polo Shirt,Blue,M"; shirts[4] = "S005,Tan Polo Shirt,Tan,XL"; shirts[5] = "S006,Black T-Shirt,Black,XL"; shirts[6] = "S007,White T-Shirt,White,XL"; shirts[7] = "S008,White T-Shirt,White,L"; shirts[8] = "S009,Green T-Shirt,Green,S"; shirts[9] = "S010,Orange T-Shirt,Orange,S"; shirts[10] = "S011,Maroon Polo Shirt,Maroon,S";
Преобразуйте строковый массив в массив класса Shirt и выведите его на консоль.

Задание 4. (30%)

Разработайте класс, который получает строковое представление телефонного номера в одном из двух возможных строковых форматов:
+<Код страны><Номер 10 цифр>, например “+79175655655” или
“+104289652211” или
8<Номер 10 цифр> для России, например “89175655655”
и преобразует полученную строку в формат:
+<Код страны><Три цифры>–<Три цифры>–<Четыре цифры>

Задание 4. (30%)

В методе main считай с консоли имя файла, который содержит слова, разделенные пробелом. В методе getLine() используя StringBuilder расставьте все слова в таком порядке, чтобы последняя буква данного слова совпадала с первой буквой следующего не учитывая регистр. Каждое слово должно участвовать 1 раз.

Контрольные вопросы

1. Классы для работы со строками: общие сведения.
2. Какие методы класса String Вы знаете.
3. Методы класса StringBuffer.

Список литературы

1. The Java Language Specification, Java SE 7 Edition [электронный документ] :
2. http://docs.oracle.com/javase/specs
3. https://javarush.ru/groups/posts/2351-znakomstvo-so-string-stringbuffer-i-stringbuilder-v-java
4. Bloch, Joshua. Effective JavaTM. Second Edition. – Addison-Wesley, 2008.
5. Хабибулин И.Ш. Java 7 // И.Ш. Хабибулин. – СПб.: БХВ-Петербург, 2012. –

768 с.: ил (В подлиннике).

Задание 1.

Разработать класс Person, в котором имеется функция, возвращающая Фамилию И.О. Функция должна учитывать возможность отсутствия значений в полях Имя и Отчество. Программу оптимизировать с точки зрения быстродействия.

Задание 2.

Доработать класс адреса, который из полученной строки формата "Страна[d] Регион[d] Город[d] Улица[d] Дом[d] Корпус[d] Квартира" ([d] – разделитель, например, «запятая») выбирает соответствующие части и записывает их в соответствующие поля класса Address. Учесть, что в начале и конце разобранной части адреса не должно быть пробелов. Все поля адреса строковые. Разработать проверочный класс не менее чем на четыре адресных строки. В программе предусмотреть две реализации этого метода:
а) разделитель – только запятая (использовать метод split()); Внимание, при разработке нужно учесть, что
б) разделитель – любой из символов „.;- (класс StringTokenizer).

Задание 3.

Реализуйте класс Shirt: Метод toString() выводит объяснение и значение полей построчно.

Дан также строковый массив: shirts[0] = "S001,Black Polo Shirt,Black,XL"; shirts[1] = "S002,Black Polo Shirt,Black,L"; shirts[2] = "S003,Blue Polo Shirt,Blue,XL"; shirts[3] = "S004,Blue Polo Shirt,Blue,M"; shirts[4] = "S005,Tan Polo Shirt,Tan,XL"; shirts[5] = "S006,Black T-Shirt,Black,XL"; shirts[6] = "S007,White T-Shirt,White,XL"; shirts[7] = "S008,White T-Shirt,White,L"; shirts[8] = "S009,Green T-Shirt,Green,S"; shirts[9] = "S010,Orange T-Shirt,Orange,S"; shirts[10] = "S011,Maroon Polo Shirt,Maroon,S";
Преобразуйте строковый массив в массив класса Shirt и выведите его на консоль.

Задание 4.

Разработайте класс, который получает строковое представление телефонного номера в одном из двух возможных строковых форматов:
+<Код страны><Номер 10 цифр>, например “+79175655655” или
“+104289652211” или
8<Номер 10 цифр> для России, например “89175655655”
и преобразует полученную строку в формат:
+<Код страны><Три цифры>–<Три цифры>–<Четыре цифры>

Задание 5.

В методе main считай с консоли имя файла, который содержит слова, разделенные пробелом. В методе getLine() используя StringBuilder расставьте все слова в таком порядке, чтобы последняя буква данного слова совпадала с первой буквой следующего не учитывая регистр. Каждое слово должно участвовать 1 раз.

Практическая работа № 16

В процессе написания тестовых заданий ознакомьтесь с принципами создания динамических структур в Java, механизмом исключений и концепцией интерфейсов.

Замечание: в процессе выполнения задания НЕЛЬЗЯ пользоваться утилитными классами Java (за исключением java.util.HashMap)

Задание на практическую работу

Задание 1

Создайте класс Drink – напитка. Класс описывает сущность – напиток и характеризуется следующими свойствами - стоимостью, названием и описанием**. Класс должен быть определен как неизменяемый (Immutable class).**

Конструкторы:
 принимающий два параметра – название и описание. Стоимость при этом инициализируется значением 0;
 принимающий три параметра – стоимость, название и описание.

Методы:

−возвращающий стоимость.

− возвращающий название.

−возвращающий описание.

***Дополнительные требования*:**

Вместо литералов в коде (магических констант) необходимо использовать константы класса, содержащие эти значения. Пояснение: в этом случае вы локализуете изменения этих значений в одном месте, а имя константы скажет нам о сути литерала. Этот класс должен быть неизменяемым (правила проектирования таких классов приводятся в лекциях).

Задание 2

Создайте интерфейс Item – для работы с позициями заказа. Интерфейс определяет 3 метода:

- возвращает стоимость.
- возвращает название.
- возвращает описание позиции.

Класс Drink и Dish должны реализовывать этот интерфейс.
Класс Dish сделайте неизменяемым (аналогично Drink). Order должен хранить (удалять и добавлять) не только экземпляры класса Dish, но и Drink (Для этого разработайте классы Order и TablesOrderManager).

Задание 3

Создайте класс InternetOrder, который моделирует сущность интернет заказ в ресторане или кафе. Класс основан на циклическом двусвязном списке с выделенной головой и может хранить как блюда, так и напитки. Внимание: список реализуется самостоятельно.

Конструкторы:

- не принимающий параметров (для списка создается только головной элемент, сам список пуст).
- принимающий массив позиций заказа (созхдаем список из N позиций.

Методы:

 добавляющий позицию в заказ (принимает ссылку типа Item). Пока этот метод возвращает истину после выполнения операции добавления элемента.
−удаляющий позицию из заказа по его названию (принимает название блюда или напитка в качестве параметра). Если позиций с заданным названием несколько, всегда удаляются последние. Возвращает логическое значение (true, если элемент был удален).

 удаляющий все позиции с заданным именем (принимает название в качестве параметра). Возвращает число удаленных элементов.

 возвращающий общее число позиций заказа (повторяющиеся тоже считаются) в заказе.

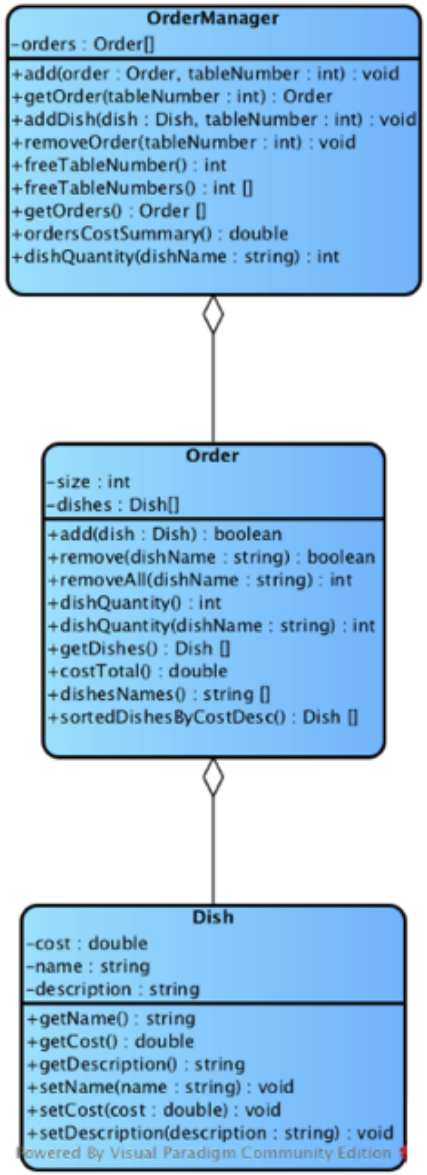
 возвращающий массив заказанных блюд и напитков (*значений null в массиве быть не должно*).

 возвращающий общую стоимость заказа.

 возвращающий число заказанных блюд или напитков (принимает название блюда или напитка в качестве параметра).

 возвращающий массив названий заказанных блюд и напитков (без повторов).

 возвращающий массив позиций заказа, отсортированный по убыванию цены. *Дополнительные требования*:


OrderManager
-orders : Order[]
+add(order : Order, tableNumber : int) : void
+getOrder(tableNumber : int) : Order
+addDish(dish : Dish, tableNumber : int) : void
+removeOrder(tableNumber : int) : void
+freeTableNumber() : int
+freeTableNumbers() : int []
+getOrders() : Order []
+ordersCostSummary() : double
+dishQuantity(dishName : string) : int
Order
-size : int
-dishes : Dish[]
+add(dish : Dish) : boolean
+remove(dishName : string) : boolean
+removeAll(dishName : string) : int
+dishQuantity() : int
+dishQuantity(dishName : string) : int
+getDishes() : Dish []
+costTotal() : double
+dishesNames() : string []
+sortedDishesByCostDesc() : Dish []
Dish
-cost : double
-name : string
-description : string
+getName() : string
+getCost() : double
+getDescription() : string
+setName(name : string) : void
+setCost(cost : double) : void
+setDescription(description : string) : void
Powered By Visual Paradigm Community Edition

Задание 4

Переименуйте класс Order из предыдущего задания в RestaurantOrder.
Создайте интерфейс Order – позиции заказа.

Интерфейс должен определять следующие методы:

 добавления позиции в заказ (принимает ссылку типа *Item*), при этом возвращает логическое значение.

  удаляет позицию из заказа по его названию (принимает название блюда или напитка в качестве параметра). Возвращает логическое значение.

удаляет все позиции с заданным именем (принимает название в качестве параметра). Возвращает число удаленных элементов.

возвращает общее число позиций заказа в заказе.

−возвращает массив позиций заказа.

−возвращает общую стоимость заказа.

возвращает число заказанных блюд или напитков (принимает название в качестве параметра).

−возвращает массив названий заказанных блюд и напитков (без повторов).

−возвращает массив позиций заказа, отсортированный по убыванию цены.

Замечание: Классы *InternetOrder* и *RestaurantOrder* должны реализовывать интерфейс *Order*.

Задание 5

Переименуйте класс *TablesOrderManager* в *OrderManager*. Добавьте ему еще одно поле типа *java.util.HasMap<String*, *Order>*, которое содержит пары адрес-заказ, и методы (работающие с этим полем):

Методы класса:

−перегрузка метода добавления заказа. В качестве параметров принимает строку – адрес и ссылку на заказ.

−перегрузка метода получения заказа. В качестве параметра принимает строку – адрес.

−перегрузка метода удаления заказа. В качестве параметра принимает строку – адрес заказа.

−перегрузка метода добавления позиции к заказу. В качестве параметра принимает адрес и *Item*.

 возвращающий массив имеющихся на данный момент интернет-заказов.

 возвращающий суммарную сумму имеющихся на данный момент интернет-заказов.

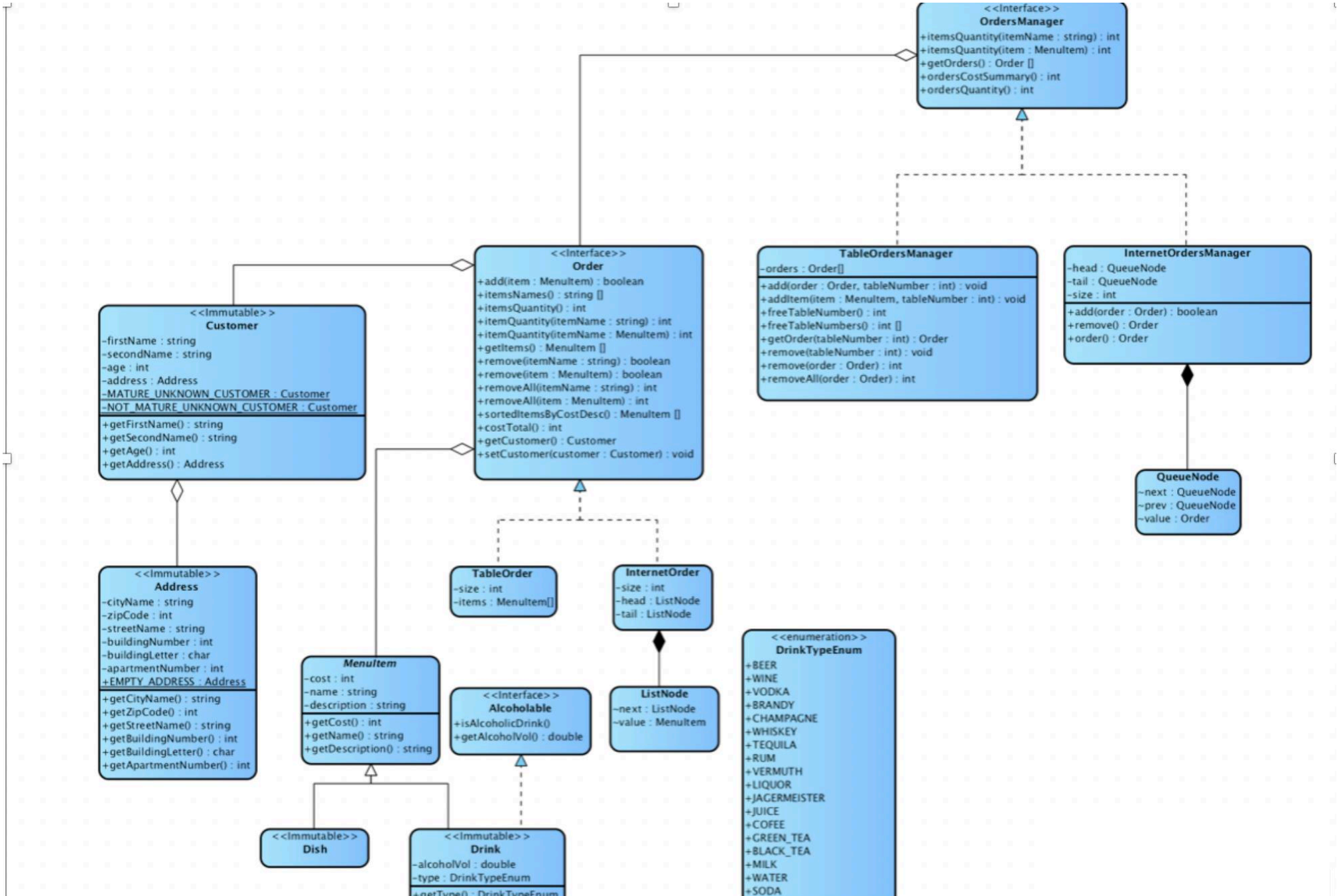
 возвращающий общее среди всех интернет-заказов количество заказанных порций заданного блюда по его имени. Принимает имя блюда в качестве параметра. Методы должны работать с интерфейсными ссылками *Order* и *Item*.

Задание 6

Создайте объявляемое исключение *OrderAlreadyAddedException*, выбрасываемое при попытке добавить заказ столику или по адресу, если со столиком или адресатом уже связан заказ.

Конструктор классов *Drink* и *Dish* должен выбрасывать исключение java.lang.IllegalArgumentException при попытке создать блюдо или напиток со стоимостью меньше 0, без имени или описания (если параметры имя и описание - пустые строки).

Создайте не объявляемое исключение *IllegalTableNumber*, выбрасываемое в методах, принимающих номер столика в качестве параметра, если столика с таким номером не существует.


<<Interface>>
OrdersManager
+itemsQuantity(itemName : string) : int
+itemsQuantity(item : MenuItem) : int
+getOrders() : Order []
+ordersCostSummary() : int
+ordersQuantity() : int
<<Interface>>
Order
+add(item : MenuItem) : boolean
+itemsNames() : string []
+itemsQuantity() : int
+itemQuantity(itemName : string) : int
+itemQuantity(itemName : MenuItem) : int
+getItems() : MenuItem []
+remove(itemName : string) : boolean
+remove(item : MenuItem) : boolean
+removeAll(itemName : string) : int
+removeAll(item : MenuItem) : int
+sortedItemsByCostDesc() : MenuItem []
+costTotal() : int
+getCustomer() : Customer
+setCustomer(customer : Customer) : void
TableOrdersManager
-orders : Order[]
+add(order : Order, tableNumber : int) : void
+addItem(item : MenuItem, tableNumber : int) : void
+freeTableNumber() : int
+freeTableNumbers() : int []
+getOrder(tableNumber : int) : Order
+remove(tableNumber : int) : void
+remove(order : Order) : int
+removeAll(order : Order) : int
InternetOrdersManager
-head : QueueNode
-tail : QueueNode
-size : int
+add(order : Order) : boolean
+remove() : Order
+order() : Order
QueueNode
~next : QueueNode
~prev : QueueNode
~value : Order
<<Immutable>>
Customer
-firstName : string
-secondName : string
-age : int
-address : Address
-MATURE_UNKNOWN_CUSTOMER : Customer
-NOT_MATURE_UNKNOWN_CUSTOMER : Customer
+getFirstName() : string
+getSecondName() : string
+getAge() : int
+getAddress() : Address
<<Immutable>>
Address
-cityName : string
-zipCode : int
-streetName : string
-buildingNumber : int
-buildingLetter : char
-apartmentNumber : int
+EMPTY_ADDRESS : Address
+getCityName() : string
+getZipCode() : int
+getStreetName() : string
+getBuildingNumber() : int
+getBuildingLetter() : char
+getApartmentNumber() : int
TableOrder
-size : int
-items : MenuItem[]
InternetOrder
-size : int
-head : ListNode
-tail : ListNode
ListNode
~next : ListNode
~value : MenuItem
MenuItem
-cost : int
-name : string
-description : string
+getCost() : int
+getName() : string
+getDescription() : string
<<Interface>>
Alcoholable
+isAlcoholicDrink()
+getAlcoholVol() : double
<<enumeration>>
DrinkTypeEnum
+BEER
+WINE
+VODKA
+BRANDY
+CHAMPAGNE
+WHISKEY
+TEQUILA
+RUM
+VERMUTH
+LIQUOR
+JAGERMEISTER
+JUICE
+COFEE
+GREEN_TEA
+BLACK_TEA
+MILK
+WATER
+SODA
<<Immutable>>
Dish
<<Immutable>>
Drink
-alcoholVol : double
-type : DrinkTypeEnum