Программирование на Java

Лекция 14. Пакет java.util

20 апреля 2003 года

Авторы документа:

Николай Вязовик (Центр Sun технологий МФТИ) <vyazovick@itc.mipt.ru>
Евгений Жилин (Центр Sun технологий МФТИ) < gene@itc.mipt.ru>

Аннотация

Эта лекция посвящена пакету java.util, в котором содержится множество вспомогательных классов и интерфейсов. Они настолько удобны, что практически любая программа использует эту библиотеку. Центральную часть в изложении занимает тема контейнеров или коллекций – классов, хранящих упорядоченные ссылки на ряд объектов. Они были серьезно переработаны в ходе создания версии Java2. Также рассматриваются классы для работы с датой, для генерации случайных чисел, для обеспечения поддержки многих национальных языков в приложении и другие.

Оглавление

Лекция 14. Пакет java.util 1
1. Введение 2
2. Работа с датами и временем 2
2.1. Класс Date 2
2.2. Классы Calendar и GregorianCalendar 2
2.3. Класс TimeZone 6
2.4. Класс SimpleTimeZone 8
3. Интерфейс Observer и класс Observable 10
4. Коллекции 11
4.1. Интерфейсы 12
4.1.1. Интерфейс Collection 12
4.1.2. Интерфейс Set 12
4.1.3. Интерфейс List 12
4.1.4. Интерфейс Map 13
4.1.5. Интерфейс SortedSet 13
4.1.6. Интерфейс SortedMap 13
4.1.7. Интерфейс Iterator 13
4.2. Абстрактные классы используемые при работе с коллекциями 14
4.3. Конкретные классы коллекций 16
4.4. Класс Collections 22
5. Класс Properties 23
6. Интерфейс Comparator 25
7. Класс Arrays 25
8. Класс StringTokenizer 26
9. Класс BitSet 26
10. Класс Random 27
11. Локализация 28
11.1. Класс Locale 28
11.2. Класс ResourceBundle 30
12. Заключение 37
13. Контрольные вопросы 38

Лекция 14. Пакет java.util

Содержание лекции.

1. Введение 2

2. Работа с датами и временем 2

2.1. Класс Date 2

2.2. Классы Calendar и GregorianCalendar 2

2.3. Класс TimeZone 6

2.4. Класс SimpleTimeZone 8

3. Интерфейс Observer и класс Observable 10

4. Коллекции 11

4.1. Интерфейсы 12

4.1.1. Интерфейс Collection 12

4.1.2. Интерфейс Set 12

4.1.3. Интерфейс List 12

4.1.4. Интерфейс Map 13

4.1.5. Интерфейс SortedSet 13

4.1.6. Интерфейс SortedMap 13

4.1.7. Интерфейс Iterator 13

4.2. Абстрактные классы используемые при работе с коллекциями 14

4.3. Конкретные классы коллекций 16

4.4. Класс Collections 22

5. Класс Properties 23

6. Интерфейс Comparator 25

7. Класс Arrays 25

8. Класс StringTokenizer 26

9. Класс BitSet 26

10. Класс Random 27

11. Локализация 28

11.1. Класс Locale 28

11.2. Класс ResourceBundle 30

12. Заключение..... 37

13. Контрольные вопросы..... 38

1. Введение

В Java имеется большое количество вспомогательных классов. Далее мы рассмотрим наиболее важные классы пакета java.util.

2. Работа с датами и временем

2.1. Класс Date

Класс Date изначально предоставлял набор функций для работы с датой - для получения текущего года, месяца и т.д. однако сейчас все эти методы не рекомендованы к использованию и практически всю функциональность для этого предоставляет класс Calendar. Класс Date так же определен в пакете java.sql поэтому желательно указывать полностью квалифицированное имя класса Date.

Существует несколько конструкторов класса Date однако рекомендовано к использованию два

Date() и Date(long date)

второй конструктор использует в качестве параметра значение типа long который указывает на количество миллисекунд прошедшее с 1 Января 1970, 00:00:00 по Гринвичу. Первый конструктор создает дату использует текущее время и дату (т.е. время выполнения конструктора). Фактически это эквивалентно второму варианту new Date(System.current-TimeMillis); Можно уже после создания экземпляра класса Date использовать метод setTime(long time), для того, что бы задать текущее время.

Для сравнения дат служат методы after(Date date), before(Date date) которые возвращают булевское значение в зависимости от того выполнено условие или нет. Метод compareTo(Date anotherDate) возвращает значение типа int которое равно -1 если дата меньше сравниваемой, 1 если больше и 0 если даты равны. Метод toString() представляет строковое представление даты, однако для форматирования даты в виде строк рекомендуется пользоваться классом SimpleDateFormat определенном в пакте java.text

2.2. Классы Calendar и GregorianCalendar

Более развитые средства для работы с датами представляет класс Calendar. Calendar является абстрактным классом. Для различных платформ реализуются конкретные подклассы календаря. На данный момент существует реализация Грегорианского календаря - GregorianCalendar. Экземпляр этого класса получается вызовом статического метода getInstance(), который возвращает экземпляр класса Gregorian. Подклассы класса Calendar должны интерпретировать объект Date по разному. В будущем предполагается реализовать так же лунный календарь, используемый в некоторых странах.

Calendar обеспечивает набор методов позволяющих манипулировать различными "частями" даты, т.е. получать и устанавливать дни, месяцы, недели и т.д.

Если при задании параметров календаря упущены некоторые параметры, то для них будут использованы значения по умолчанию для начала отсчета. т.е.

YEAR = 1970, MONTH = JANUARY, DATE = 1 и т.д.

Для считывания, установки манипуляции различных "частей" даты используются методы get(int filed), set(int filed, int value), add(int field, int amount), roll(int field, inr amount), переменная типа int с именем filed указывает на номер поля с которым нужно произвести операцию. Для удобства все эти поля определены в Calendar, как статические константы типа int.

Рассмотрим подробнее порядок выполнения перечисленных методов.

set(int field,int value)

Как уже отмечалось ранее данный метод производит установку какого - либо поля даты. На самом деле после вызова этого метода, немедленного пересчета даты не производится. Пересчет даты будет осуществлен только после вызова методов get(), getTime() или TimeInMillis(). Т.о. последовательная установка нескольких полей, не вызовет не нужных вычислений. Помимо этого появляется еще один интересный эффект. Рассмотрим следующий пример. Предположим, что дата установлена на последний день августа. Необходимо перевести ее на последний день сентября. Если внутреннее представление даты изменялось бы после вызова метода set, то при последовательной установке полей мы получили бы вот такой эффект.

```
public class Test {

    public Test() {
    }
    public static void main(String[] args) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMMM dd HH:mm:ss");
         Calendar cal = Calendar.getInstance();
         cal.set(Calendar.YEAR,2002);
         cal.set(Calendar.MONTH,Calendar.AUGUST);
         cal.set(Calendar.DAY_OF_MONTH,31);
         System.out.println(" Initialy set date:        " +
sdf.format(cal.getTime()));
         cal.set(Calendar.MONTH,Calendar.SEPTEMBER);
         System.out.println(" Date with month changed : " +
sdf.format(cal.getTime()));
         cal.set(Calendar.DAY_OF_MONTH,30);
         System.out.println(" Date with day changed :   " +
sdf.format(cal.getTime()));

    }
}
```

```
 Initialy set date:        2002 August 31 22:57:47
 Date with month changed : 2002 October 01 22:57:47
 Date with day changed :   2002 October 30 22:57:47
```

Как видно, в данном примере, при изменении месяца день месяца остался неизменным и было унаследовано его предыдущее значение. Но т.к. в сентябре 30 дней, то дата автоматически была переведена на 1 октября, и, когда было установлено 30 число, то оно относилось бы уже к октябрю месяцу. В следующем примере считывание даты не производится, соответственно ее вычисление не производится, до тех пор пока все поля не установлены.

```
public class Test {

    public Test() {
    }
    public static void main(String[] args) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMMM dd HH:mm:ss");
         Calendar cal = Calendar.getInstance();
         cal.set(Calendar.YEAR,2002);
         cal.set(Calendar.MONTH,Calendar.AUGUST);
         cal.set(Calendar.DAY_OF_MONTH,31);
         System.out.println(" Initialy set date:                  " +
sdf.format(cal.getTime()));
         cal.set(Calendar.MONTH,Calendar.SEPTEMBER);
         cal.set(Calendar.DAY_OF_MONTH,30);
         System.out.println(" Date with day and month changed :   " +
sdf.format(cal.getTime()));

    }
}
```

```
 Initialy set date:                  2002 August 31 23:03:51
 Date with day and month changed :   2002 September 30 23:03:51
```

add(int field,int delta)

Добавляет некоторое смещение к существующей величине поля. В принципе то же самое можно сделать с помощью set(f, get(f) + delta)

В случае использования метода add следует помнить о двух правилах.

1. Если величина поля изменения выходит за диапазон возможных значений данного поля, то производится деление по модулю данной величины, частное суммируется со следующим по старшинству полем.

2. Если изменяется одно из полей, при этом после изменения младшее по отношению к изменяемому полю, принимает некорректное значение, то оно изменяется, на, которое максимально близко к "старому".

```
public class Test {

    public Test() {
    }
```

```
    public static void main(String[] args) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMMM dd HH:mm:ss");
         Calendar cal = Calendar.getInstance();
         cal.set(Calendar.YEAR,2002);
         cal.set(Calendar.MONTH,Calendar.AUGUST);
         cal.set(Calendar.DAY_OF_MONTH,31);
         cal.set(Calendar.HOUR_OF_DAY,19);
         cal.set(Calendar.MINUTE,30);
         cal.set(Calendar.SECOND,00);
         System.out.println("Current date: " + sdf.format(cal.getTime()));
         cal.add(Calendar.SECOND,75);
         System.out.println("Current date: " + sdf.format(cal.getTime()));
         cal.add(Calendar.MONTH,1);
         System.out.println("Current date: " + sdf.format(cal.getTime()));
    }
}
```

```
Current date: 2002 August 31 19:30:00
Rule 1: 2002 August 31 19:31:15
Rule 2: 2002 September 30 19:31:15
```

roll(int field,int delta)

Добавляет некоторое смещение к существующей величине поля и не производит изменения старших полей. Рассмотрим приведенный ранее пример, но с использованием метода roll

```
public class Test {

    public Test() {
    }
    public static void main(String[] args) {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMMM dd HH:mm:ss");
         Calendar cal = Calendar.getInstance();
         cal.set(Calendar.YEAR,2002);
         cal.set(Calendar.MONTH,Calendar.AUGUST);
         cal.set(Calendar.DAY_OF_MONTH,31);
         cal.set(Calendar.HOUR_OF_DAY,19);
         cal.set(Calendar.MINUTE,30);
         cal.set(Calendar.SECOND,00);
         System.out.println("Current date: " + sdf.format(cal.getTime()));
         cal.roll(Calendar.SECOND,75);
         System.out.println("Rule 1: " + sdf.format(cal.getTime()));
         cal.roll(Calendar.MONTH,1);
         System.out.println("Rule 2: " + sdf.format(cal.getTime()));
    }
}
```

```
Current date: 2002 August 31 19:30:00
```

```
Rule 1: 2002 August 31 19:30:15
Rule 2: 2002 September 30 19:30:15
```

Как видно из результатов работы приведенного выше кода, действие правила 1 изменилось, по сравнению с методом add, а правило 2 действует так же.

2.3. Класс TimeZone

Класс TimeZone предназначен для совместного использования с классами Calendar и DateFormat. Класс абстрактный, поэтому нельзя создать конкретный экземпляр этого с помощью конструктора. Для этого определен статический метод getDefault(), который возвращает экземпляр класса TimeZone с настройками взятыми из настроек операционной системы под управлением которой работает JVM. Для того, что бы получить экземпляр TimeZone с конкретными настройками, можно воспользоваться статическим методом getTimeZone(String ID), в качестве параметра, которому передается наименование конкретного временного пояса, для которого необходимо получить объект TimeZone. Нигде не определено публичного набора полей определяющих возможный набор параметров для getTimeZone. Вместо этого определен статический метод String[] getAvailableIds() который возвращает массив строк с возможными параметрами для getTimeZone. Можно так определить набор возможных параметров для конкретного временного пояса (рассчитывается относительно Гринвича) String[] getAvailableIds(int offset);

Рассмотрим пример в котором на консоль последовательно выводятся

- временная зона по умолчанию
- список всех возможных временных зон
- список временных зон которые совпадают с текущей временной зоной.

Следует обратить внимание что на консоль выводится полное наименование временной зоны. Именно в таком формате следует передавать параметр для getTimeZone(). Хотя и допускается указание строки, которую возвращает метод getDisplayName(), так же возможно указание временного пояса в формате "GMT-8:00".

```
public class Test  {
    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
        TimeZone tz = TimeZone.getDefault();
        int rawOffset = tz.getRawOffset();
        System.out.println("Current TimeZone" + tz.getDisplayName() +
tz.getID() + "\n\n");

        // Dispaly all available TimeZones
        System.out.println("All Available TimeZones \n");
        String[] idArr = tz.getAvailableIDs();
        for(int cnt=0;cnt < idArr.length;cnt++){
            tz = TimeZone.getTimeZone(idArr[cnt]);
           System.out.println(test.padr(tz.getDisplayName() + tz.getID(),64)
```

```
 + "  raw offset=" + tz.getRawOffset() + ";hour offset=(" + tz.getRawOffset()/
 (1000 * 60 * 60 ) + ")");
        }

        // Dispaly all available TimeZones same as for Moscow
        System.out.println("\n\n TimeZones same as for Moscow \n");
        idArr = tz.getAvailableIDs(rawOffset);
        for(int cnt=0;cnt < idArr.length;cnt++){
            tz = TimeZone.getTimeZone(idArr[cnt]);
            System.out.println(test.padr(tz.getDisplayName()+ tz.getID(),64)
 + "  raw offset=" + tz.getRawOffset() + ";hour offset=(" + tz.getRawOffset()/
 (1000 * 60 * 60 ) + ")");
        }

    }

    String padr(String str,int len){
        if(len - str.length() > 0){
            char[] buf = new char[len - str.length()];
            Arrays.fill(buf,' ');
            return str + new String(buf);
        }else{
            return str.substring(0,len);
        }

    }
}
```

```
Current TimeZone Moscow Standard TimeEurope/Moscow

All Available TimeZones
```

… полный список временных зон приведен в приложении XXX …

TimeZones same as for Moscow

```
Eastern African TimeAfrica/Addis_Aba  raw offset=10800000;hour offset=(3)
Eastern African TimeAfrica/Asmera     raw offset=10800000;hour offset=(3)
Eastern African TimeAfrica/Dar_es_Sa  raw offset=10800000;hour offset=(3)
Eastern African TimeAfrica/Djibouti   raw offset=10800000;hour offset=(3)
Eastern African TimeAfrica/Kampala    raw offset=10800000;hour offset=(3)
Eastern African TimeAfrica/Khartoum   raw offset=10800000;hour offset=(3)
Eastern African TimeAfrica/Mogadishu  raw offset=10800000;hour offset=(3)
Eastern African TimeAfrica/Nairobi    raw offset=10800000;hour offset=(3)
Arabia Standard TimeAsia/Aden         raw offset=10800000;hour offset=(3)
Arabia Standard TimeAsia/Baghdad      raw offset=10800000;hour offset=(3)
Arabia Standard TimeAsia/Bahrain      raw offset=10800000;hour offset=(3)
Arabia Standard TimeAsia/Kuwait       raw offset=10800000;hour offset=(3)
Arabia Standard TimeAsia/Qatar        raw offset=10800000;hour offset=(3)
```

```
Arabia Standard TimeAsia/Riyadh       raw offset=10800000;hour offset=(3)
Eastern African TimeEAT               raw offset=10800000;hour offset=(3)
Moscow Standard TimeEurope/Moscow     raw offset=10800000;hour offset=(3)
Eastern African TimeIndian/Antananar  raw offset=10800000;hour offset=(3)
Eastern African TimeIndian/Comoro     raw offset=10800000;hour offset=(3)
Eastern African TimeIndian/Mayotte    raw offset=10800000;hour offset=(3)
```

2.4. Класс SimpleTimeZone

Класс SimpleTimeZone являясь потомком TimeZone реализует его абстрактные методы и предназначен для использования в настройках использующих Григорианский календарь. В большинстве случаев нет необходимости создавать экземпляр этого класса с помощью конструктора. Вместо этого лучше использовать статические методы которые возвращают экземпляр класса TimeZone рассмотренные в предыдущем параграфе. Единственная, пожалуй, причина для использования конструктора - необходимость задания нестандартных правил перехода на зимнее и летнее время.

В классе SimpleTimeZone определено три конструктора. Рассмотрим наиболее полный, с точки зрения функциональности) вариант, который помимо временной зоны задает летнее и зимнее время.

```
public SimpleTimeZone(int rawOffset,
                      String ID,
                      int startMonth,
                      int startDay,
                      int startDayOfWeek,
                      int startTime,
                      int endMonth,
                      int endDay,
                      int endDayOfWeek,
                int endTime)
```

rawOffset - временное смещение относительно гринвича

ID - идентификатор временной зоны. (см. пред.параграф)

startMonth - месяц перехода на летнее время

startDay - день месяца перехода на летнее время*

startDayOfWeek - день недели перехода на летнее время*

startTime - время перехода на летнее время (указывается в миллисекундах)

endMonth - месяц окончания действия летнего времени

endDay - день окончания действия летнего времени*

endDayOfWeek - день недели окончания действия летнего времени*

endTime - время окончания действия летнего времени (указывается в миллисекундах)

Месяц перехода времени начинает отсчитываться с нуля. Для этих целей можно применять константы определенные в классе Calendar, например Calendar.JANUARY

Перевод часов на зимний и летний вариант исчисления времени определяется специальным правительственным указом. Обычно переход на летнее время происходит в 2 часа в последнее воскресенье марта, а переход на зимнее время - в 3 часа в последнее воскресенье октября.

Алгоритм расчета таков

- если startDay 1 и установлен день недели, то будет вычисляться первый день недели startDayOfWeek месяца startMonth (например первое воскресенье)
- если startDay -1, и установлен день недели, то будет вычисляться последний день недели startDayOfWeek месяца startMonth (например последнее воскресенье)
- если день недели startDayOfWeek установлен в 0, то будет вычисляться число startDay конкретного месяца startMonth
- для того что бы установить день недели после конкретного числа специфицируется отрицательное значение дня недели. Например, что бы указать первый понедельник после 23 февраля используется вот такой набор startDayOfWeek=-MONDAY, start-Month=FEBRUARY, startDay=23
- для того что бы указать последний день недели перед каким-либо числом, указывается отрицательное значение этого числа и отрицательное значение дня недели. Например, для того что бы указать последнюю субботу перед 23 февраля необходимо задать такой набор параметров startDayOfWeek=- -SATURDAY, startMonth=FEBRUARY, startDay=-23
- все вышеперечисленное относится так же и к окончанию действия летнего времени.

Рассмотрим пример получения текущей временной зоны с заданием перехода на зимнее и летнее время для России по умолчанию.

```
public class Test  {
    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
        SimpleTimeZone stz = new SimpleTimeZone(
                TimeZone.getDefault().getRawOffset()
               ,TimeZone.getDefault().getID()
               ,Calendar.MARCH
               ,-1
               ,Calendar.SUNDAY
               ,test.getTime(2,0,0,0)
               ,Calendar.OCTOBER
               ,-1
               ,Calendar.SUNDAY
               ,test.getTime(3,0,0,0)
                );
        System.out.println(stz.toString());
    }
    int getTime(int hour,int min,int sec,int ms){
        return hour * 3600000 + min * 60000 + sec * 1000 + ms;
```

```
    }
}
```

```
java.util.SimpleTimeZone[id=Europe/Moscow,offset=10800000,dstSavings=3600000,useDaylight=true,startYear=
0,startMode=2,startMonth=2,startDay=-1,startDayOfWeek=1,startTime=7200000,startTimeMode=0,endMode=
2,endMonth=9,endDay=-1,endDayOfWeek=1,endTime=10800000,endTimeMode=0]
```

3. Интерфейс Observer и класс Observable

Интерфейс Observable определяет всего один метод update(Observable o, Object arg), который вызывается когда обозреваемый объект изменяется.

Класс Observer предназначен для поддержки обозреваемого объекта в парадигме MVC (model-view-controller), которая, как и другие проектные решения и шаблоны, рассмотрена в специальной литературе. Этот класса должен быть унаследован, если возникает необходимость в том, отслеживать состояние какого - либо объекта. Обозреваемый объект может иметь несколько обозревателей. Соответственно они должны реализовать интерфейс Observable.

После того как в состоянии обозреваемого объекта что-то меняется, то необходимо вызвать метод notifyObservers, который в свою очередь вызывает методы update у каждого обозревателя.

Порядок в котором вызываются методы update обозревателей заранее не определен. Реализация по умолчанию подразумевает их вызов в порядке регистрации. Регистрация осуществляется с помощью метода addObserver(Observer o); Удаление обозревателя из списка осуществляется с помощью deleteObserver(Observer o). Перед вызовом notifyObservers, необходимо вызвать метод setChanged, который устанавливает признак того, что обозреваемый объект был изменен.

Рассмотрим пример организации взаимодействия классов.

```
public class TestObservable extends java.util.Observable {
    private String name = "";
    public TestObservable(String name) {
        this.name = name;
    }

    public void modify(){
        setChanged();
    }

    public String getName(){
        return name;
    }
}
public class TestObserver  implements java.util.Observer{
    private String name = "";
```

```
    public TestObserver(String name) {
        this.name = name;
    }

    public void update(java.util.Observable o,Object arg){
        String str = "Called update of " + name;
        str += " from " + ((TestObservable)o).getName();
        str += " with argument " + (String)arg;
        System.out.println(str);
    }
}

public class Test {

    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
        TestObservable to = new TestObservable("Observable");
        TestObserver o1 = new TestObserver("Observer 1");
        TestObserver o2 = new TestObserver("Observer 2");
        to.addObserver(o1);
        to.addObserver(o2);
        to.modify();
        to.notifyObservers("Notify argument");
    }
}
```

В результате работы на консоль будет выведено.

```
Called update of Observer 2 from Observable with argument Notify argument
Called update of Observer 1 from Observable with argument Notify argument
```

На практике использовать Observer не всегда удобно, т.к. в Java отсутствует множественное наследование, и Observer необходимо наследовать в самом начале построения иерархии классов. В качестве варианта, можно предложить определить интерфейс, задающий функциональность, сходную с Observer, и реализовать его в нужном вам классе.

4. Коллекции

Зачастую в программе необходимо сгруппировать объекты в некую логическую структуру , определение которой производится во время исполнения. Наиболее простой способ сделать это с помощью массивов. Однако, несмотря на то, что это достаточно эффективное решение для многих случаев, оно имеет и ограничения. Так в массиве возможно обращение к его элементу только по его номеру (индексу). Так же необходимо знать количеств объектов организуемых в массив до его создания. Массивы существовали в Java изначально, было определено так же два класса для организации боле эффективной работы с наборами объектов Hashtable и Vector. В JDK 1.2 набор классов поддерживающих работу с коллекциями был существенно расширен.

Следует обратить внимание, что коллекции предназначены для работы с объектами. В то время как, массивы могут содержать как простые типы, так и ссылки на объекты, то классы коллекций содержат только ссылки на объекты. Однако если возникает необходимость использования простых типов в коллекциях, то необходимо использовать для этого классы-обертки.

Существует несколько различных типов классов-коллекций. Все они разработаны, по возможности следуя единой логике и определенным интерфейсам, и там где это возможно, манипулирование ими унифицировано. Однако все коллекции отличаются внутренними механизмами хранения, скоростью доступа к элементам, потребляемой памятью и другими деталями. Например в некоторых коллекциях объекты (так же называемые элементами коллекций), могут быть упорядочены, в некоторых нет. В некоторых типах коллекций допускается дублирование ссылок на объект в нет. Далее мы рассмотрим каждый из классов - коллекций

Классы обеспечивающие манипулирование коллекциями объектов, сведены в пакет java.util

4.1. Интерфейсы

4.1.1. Интерфейс Collection

Является корнем всей иерархии классов-коллекций. Он определяет базовую функциональность любой коллекции - набор методов которые позволяют добавлять, удалять, выбирать элементы коллекции. Классы которые имплементируют интерфейс Collection, могут содержать дубликаты и пустые (null) значения.

AbstractCollection, являясь абстрактным классом обеспечивает, служит основой для создания конкретных классов коллекций и содержит реализацию некоторых методов определенных в интерфейсе Collection.

4.1.2. Интерфейс Set

Классы которые реализуют этот интерфейс не разрешают наличие дубликатов. В коллекции этого типа допускается наличие только одной ссылки типа null. Интерфейс Set расширяет интерфейс Collection т.о. любой класс имплементирующий Set реализует все методы определенные в Collection. Любой объект добавляемый в Set должен реализовать метод equals для того, что бы его можно было сравнить с другими.

AbstractSet являясь абстрактным классом представляет из себя основу для реализации различных вариантов интерфейса Set

4.1.3. Интерфейс List

Классы которые реализуют этот интерфейс содержат упорядоченную последовательность объектов (Объекты хранятся в том порядке в котором они были добавлены). D JDK 1.2 был переделан класс Vector, так, что он теперь реализует интерфейс List. Интерфейс List расширяет интерфейс Collection т.о. любой класс имплементирующий List реализует все методы определенные в Collection, в то же время вводятся новые методы которые позволяют добавлять и удалять элементы из списка. List обеспечивает так же ListIterator который позволяет перемещаться как вперед, так и назад, по элементам списка.

AbstractList являясь абстрактным классом представляет из себя основу для реализации различных вариантов интерфейса List

Следует помнить, что в пакете java.awt так же определен класс List используемый для представления списка вариантов. Т.о. что бы избежать путаницы, следует использовать полностью квалифицированное имя java.util.List

4.1.4. Интерфейс Map

Классы которые реализуют этот интерфейс хранят набор неупорядоченный набор объектов парами ключ/значение. Каждый ключ должен быть уникальным. Hashtable после модификации в JDK 1.2 реализует интерфейс Map. Порядок следования пар ключ/значение не определен.

Интерфейс Map не расширяет интерфейс Collection. AbstractMap являясь абстрактным классом представляет из себя основу для реализации различных вариантов интерфейса Map

Следует обратить внимание, что List и Set расширяют интерфейс Collection, а Map нет.

4.1.5. Интерфейс SortedSet

Этот интерфейс расширяет Set требуя, что бы содержимое набора было упорядочено. Только объект имплементирующие SortedSet могут содержать объекты которые реализуют интерфейс Comparator или могут сравниваться с использованием внешнего объекта реализующего интерфейс comparator.

4.1.6. Интерфейс SortedMap

Этот интерфейс расширяет Map требуя, что бы содержимое коллекции было упорядочено по значениям ключей.

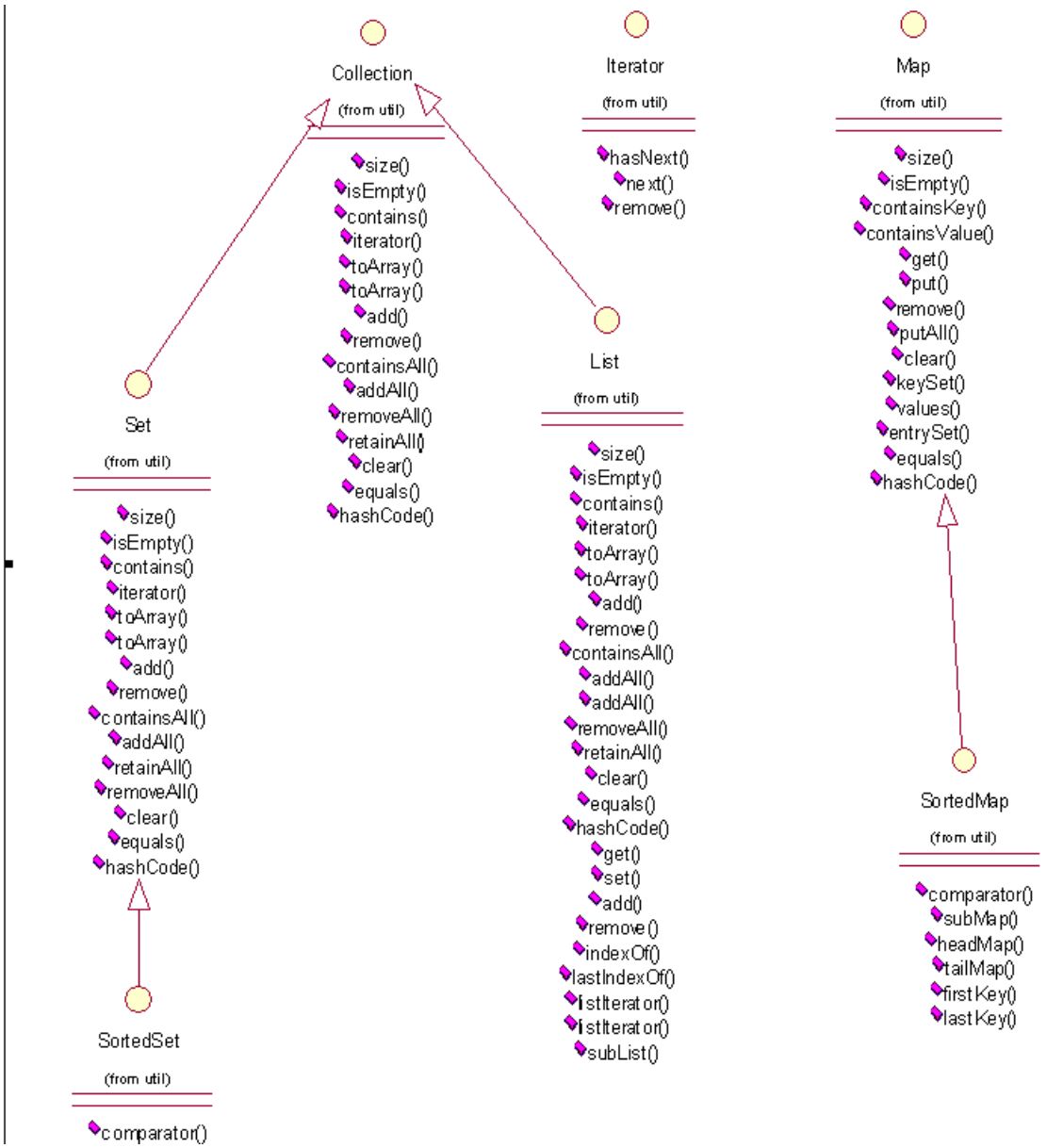
4.1.7. Интерфейс Iterator

В Java 1 для перебора элементов коллекции использовался интерфейс Enumeration. В Java 2 для этих целей должны использоваться объекты которые реализуют интерфейс Iterator. Все классы которые реализуют интерфейс Collection должны реализовать метод, iterator, который возвращает объект реализующий интерфейс Iterator. Iterator весьма похож на Enumeration, с тем лишь отличием, что в нем определен метод remove, который позволяет удалить объект из коллекции, для которой Iterator бы создан.

Т.о подводя итог перечислим:

Интерфейсы используемые при работе с коллекциями.

```
java.util.Collection
java.util.Set
java.util.List
java.util.Map
java.util.SortedSet
java.util.SortedMap
java.util.Iterator
```



4.2. Абстрактные классы используемые при работе с коллекциями.

java.util.AbstractCollection - этот класс реализует все методы определенные в интерфейсе Collection за исключением iterator и size, т.о. для того что бы создать не модифицируемую коллекцию нужно переопределить эти методы.Для реализации модифицируемой коллекции, необходимо еще переопределить метод public void add(Object o) (в противном случае, при его вызове будет возбуждено исключение UnsupportedOperationException).

Необходимо так же определить два конструктора без аргументов и с аргументом Collection. Первый должен создавать пустую коллекцию, второй коллекцию на основе сеуществующей. Данный класс расширяется классами AbstractList и AbstractSet.

java.util.AbstractList - этот класс расширяет AbstractCollection и реализует интерфейс List. Для реализации создания не модифицируемого списка необходимо имплементировать

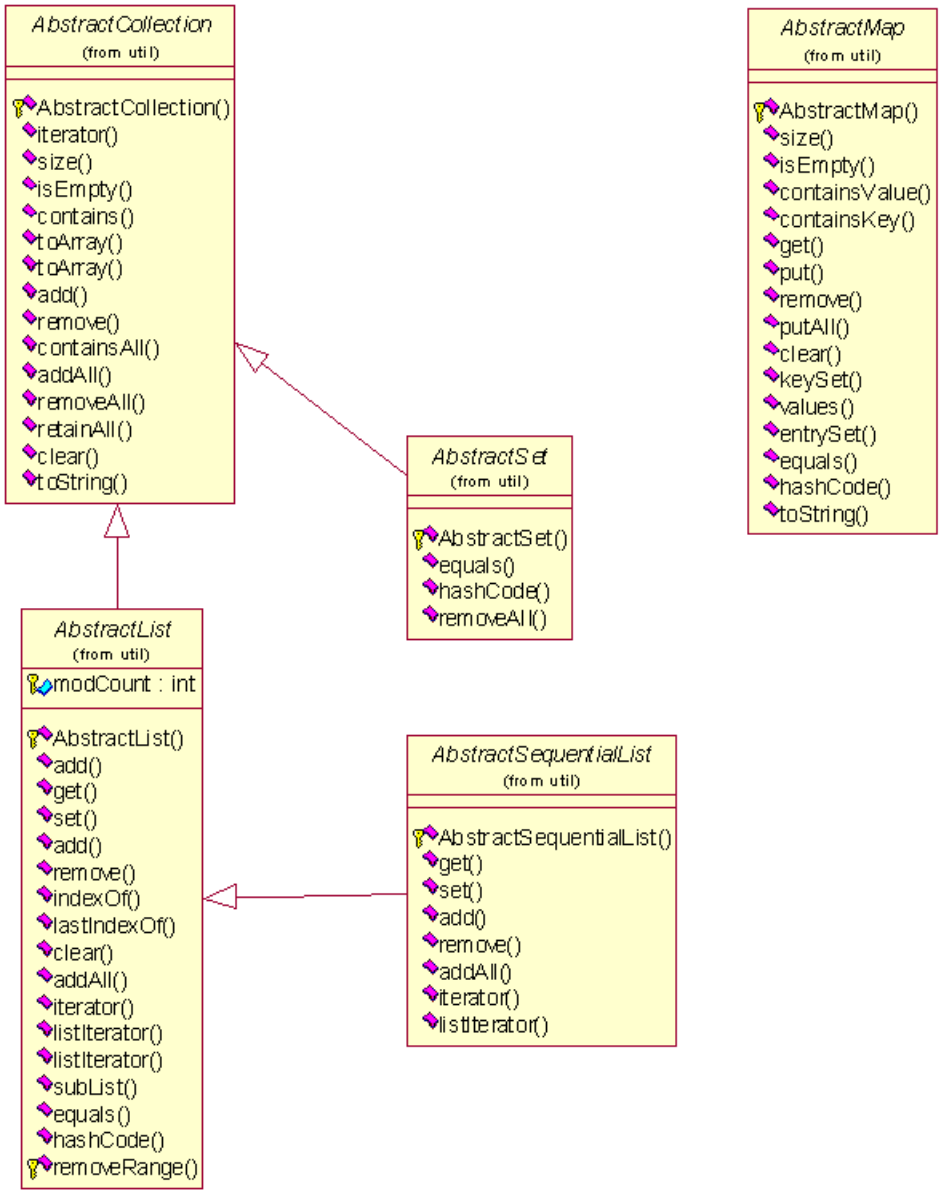
методы public Object get(int index) и public int size(). Для реализации модифицируемого списка необходимо так же реализовать метод public void set(int index,Object element); (в противном случае, при его вызове будет возбуждено исключение UnsupportedOperationException)

В отличии от AbstractCollection в этом случае нет необходимости реализовывать метод iterator, т.к. он уже реализован поверх методов доступа к элементам списка get, set, add, remove.

java.util.AbstratctSet - этот класс расширяет AbstractCollection и реализует основную функциональность определенную в интерфейсе Set. Следует отметить, что этот класс не переопределяет функциональность реализованную в классе AbstractCollection.

java.util.AbstratctMap - этот класс расширяет реализует основную функциональность определенную в интерфейсе Map Для реализации не модифицируемого класса, унаследованного от AbstractMap достаточно определить метод entrySet, который должен возвращать объект приводимый к типу AbstractSet. Этот набор (Set) не должен обеспечивать методов для добавления у даления элементов из набора. Для реализации модифицируемого класса Map необходимо так же переопределить метод put и итератор возвращаемый entrySet().iterator()

java.util.AbstratctSequentialList - этот класс расширяет AbstractList и является основой для класса LinkedList. Основное отличие от AbstractList заключается в том, что этот класс обеспечивает не только последовательный, но и произвольный доступ к элементам списка, с помощью методов get(int index), set(int index, Object element), set(int index, Object element), add(int index, Object element) и remove(int index). Для того что бы реализовать данный класс необходимо переопределить методы listIterator и size. Причем если реализуется не модифицируемый список, для итератора достаточно реализовать методы hasNext, next, hasPrevious, previous и index. Для модифицируемого списка необходимо дополнительно реализовать метод set, а для списков переменной длинны еще и add и remove.



4.3. Конкретные классы коллекций

java.util.ArrayList - этот класс расширяет AbstractList и весьма похож на класс Vector. Он так же динамически расширяется как Vector, однако его методы не являются синхронизированными, в следствие чего операции с ним выполняются быстрее. Для того, что бы воспользоваться синхронизированной версией ArrayList, можно применить вот такую конструкцию

```
List l = Collections.synchronizedList(new ArrayList(...));


public class Test {
    public Test() {
```

```
    }

    public static void main(String[] args) {
        Test t = new Test();
        ArrayList al = new ArrayList();
        al.add("Firts element");
        al.add("Second element");
        al.add("Third element");
        Iterator it = al.iterator();
        while(it.hasNext()){
            System.out.println((String)it.next());
        }
        System.out.println("\n");
        al.add(2,"Insertion");
        it = al.iterator();
        while(it.hasNext()){
            System.out.println((String)it.next());
        }
    }
}
```

```
Firts element
Second element
Third element

Firts element
Second element
Insertion
Third element
```

java.util.LinkedList - является реализацией интерфейса List. Он реализует все методы интерфейса List, помимо этого добавляются еще новее методов, которые позволяют добавлять, удалять и получать элементы в конце и начале списка. LinkedList является двухсвязным списком и позволяет перемещаться как от начала в конец списка, так и наоборот.LinkedList удобно использовать для организации стека.

```
public class Test {

    public Test() {
    }

    public static void main(String[] args) {
        Test test = new Test();
        LinkedList ll = new LinkedList();

        ll.add("Element1");
        ll.addFirst("Element2");
        ll.addFirst("Element3");
```

```
        ll.addLast("Element4");
        test.dumpList(ll);

        ll.remove(2);
        test.dumpList(ll);

        String element = (String)ll.getLast();
        ll.remove(element);
        test.dumpList(ll);
    }
    private void dumpList(List list){
        Iterator it = list.iterator();
        System.out.println();
        while(it.hasNext()){
            System.out.println((String)it.next());
        }
    }
}
Element3
Element2
Element1
Element4

Element3
Element2
Element4

Element3
Element2
```

Классы LinkedList и ArrayList имеют схожую функциональность. Однако с точки зрения производительности он отличаются. Так в ArrayList заметно быстрей (примерно на порядок) осуществляются операции прохода по всему списку (итерации) и получения данных. LinkedList почти на порядок быстрее осуществляет операции удаления и добавления новых элеемнтов.

java.util.Hastable - расширяет абстрактный класс Dictionary. D JDK 1.2, класс Hashtable так же реализует интерфейс Map. Hashtable предназначен для хранения объектов в виде пар ключ/значение. Из самого названия следует, что Hashtable использует алгоритм хэшированя для увеличения скорости доступа к данным. Для того что бы выяснить принципы работы данного алгоритма рассмотрим несколько примеров.

Предположим имеется массив строк содержащий названия городов. Для того что бы найти элемент массива содержащий название города, в общем случае необходимо просмотреть весь массив, а если необходимо найти все элементы массива, то для поиска каждого, в среднем потребуется просматривать половину массива. Такой подход может оказать приемлемым только для небольших массивов .

Как уже отмечалось ранее, для того что бы увеличить скорость поиска, используется алгоритм хэширования. Каждый объект в Java унаследован от Object. Как уже отмечалось ранее, Object определено целое число которое уникально идентифицирует экземпляр

класса Object и, соответственно все экземпляры классов унаследованных от Object. Это число возвращает метод hashCode(). Именно это число и используется при сохранении ключа в Hashtable, следующим образом: разделив длину массива предназначенного для хранения ключей на код, получается некое целое число которое служит индексом для хранения ключа в массиве. array.length % hashCode()

Далее, если необходимо добавить новую пару ключ/значение вычисляется новый индекс, если этот индекс совпадает, с уже имеющимся, то создается список ключей, на которой указывает элемент массива ключей. Таким образом, при обратом извлечении ключа, необходимо вычислить индекс массива по тому же алгоритму и получить его. Если ключ в массиве единственный, то используется значение элемента массива, если хранится несколько ключей, то необходимо обойти список и выбрать нужный.

Есть несколько соображений, относящихся к производительности классов, использующих для хранения данных алгоритм хэширования. Размер массива. Если массив будет слишком мал, то связанные списки будут слишком длинными, и скорость поиска будет существенно снижаться, т.к. просмотр элементов списка будет такой же как в обычном массиве Что бы избежать этого задается некий коэффициент заполнения. При заполнении элементов массива в котором хранятся ключи (или списки ключей) на эту величину, происходит увеличение массива и производится повторное реиндексирование. Таким образом если массив будет слишком мал, то он будет быстро заполнятся и будет производится операция повторного индексирования, которая отнимает достаточно много ресурсов. С другой стороны, если массив сделать большим, то при необходимости просмотреть последовательно все элементы коллекции использующей алгоритм хэширования будет необходимо обработать большое количество пустых элементов массива ключей.

Начальный размер массива и коэффициент загрузки коллекции задаются при конструировании.

Например Hashtable ht = new Hashtable(1000,0.60);

Существует так же конструктор без параметров. который использует значения по умолчанию 101 для размера массива и 0.75 для коэффициента загрузки.

Использование алгоритма хэширования позволяет гарантировать, что скорость доступа к элементам коллекции такого типа будет увеличиваться не линейно, а логарифмически. Таким образом, при частом поиске каких либо значений по ключу имеет смысли использовать коллекции использующие алгоритм хэштрования.

java.util.HashMap,- этот класс расширяет AbstractMap и весьма похож на класс Hashtable. HashMap предназанчен для хранения пар объектов ключ/значение. Как для ключей, так для элементов допускаются значения типа null. Порядок хранения элементов в этой коллекции не совпадает с порядком их добавления. Порядок элементов в коллекции так же может меняться во времени. HashMap обеспечивает постоянное время доступа для операций get и put.

Итерация по всем элементам коллекции пропорциональна ее емкости. Поэтому имеет смысл не устанавливать размер коллекций чрезмерно большим, если достаточно часто придется осуществлять итерацию по элементам.

Методы HashMap не являются синхронизированными. Для того, что бы обеспечить нормальную работу в много потоковом варианте следует использовать либо внешнюю синхронизацию потоков, либо использовать синхронизированный вариант коллекции

Следует еще обратить внимание на разницу между HashMap и Hashtable. Hashtable существует в Java еще с первых релизов . HashMap появился в JDK 1.2. Главное отличие в том, что Hashtable не позволяет сохранять пустые значения, в HashMap это делать можно. Кроме того, методы в Hashtable являются синхронизированными, а в HasMap нет. Кроме этого, следует помнить, что начиная с JDK 1.2 Hashtable реализует интерфейс Map, что может вызвать некоторые проблемы при попытке запус ка приложений использующих более ранние версии JDK.

```
public class Test {

    private class TestObject{
        String text = "";
        public TestObject(String text){
            this.text = text;
        };
        public String getText(){
            return this.text;
        }
        public void setText(String text){
            this.text =  text;
        }
    }

    public Test() {
    }

    public static void main(String[] args) {
        Test t = new Test();
        TestObject to = null;
        HashMap hm = new HashMap();
        hm.put("Key1",t.new TestObject("Value 1"));
        hm.put("Key2",t.new TestObject("Value 2"));
        hm.put("Key3",t.new TestObject("Value 3"));

        to = (TestObject)hm.get("Key1");
       System.out.println("Object value for Key1 = " + to.getText() + "\n");

        System.out.println("Iteration over entrySet");
        Map.Entry entry = null;
       Iterator it = hm.entrySet().iterator(); // Итератор для перебора всех
 точек входа в Map
        while(it.hasNext()){
            entry = (Map.Entry)it.next();
            System.out.println("For key = " + entry.getKey() + " value = " +
 ((TestObject)entry.getValue()).getText());
        }
        System.out.println();
```

```
        System.out.println("Iteration over keySet");
        String key = "";
        it = hm.keySet().iterator(); // Итератор для перебора всех ключей в
Map
        while(it.hasNext()){
            key = (String)it.next();
            System.out.println( "For key = " + key + " value = " +
((TestObject)hm.get(key)).getText());
        }

    }
}

Object value for Key1 = Value 1

Iteration over entrySet
For key = Key3 value = Value 3
For key = Key2 value = Value 2
For key = Key1 value = Value 1

Iteration over keySet
For key = Key3 value = Value 3
For key = Key2 value = Value 2
For key = Key1 value = Value 1
```

java.util.TreeMap - расширяет класс AbstractMap и реализует интерфейс SortedMap.TreeMap содержит ключи в порядке возрастания. Используется либо натуральное сравнение ключей, либо должен быть реализован интерфейс Comparable. Реализация алгоритма поиска обеспечивает логарифмическую зависимость времени выполнения основных операций (containsKey, get, put и remove). Запрещено использование null значений для ключей. При использовании дубликатов ключей ссылка на объект сохраненный с таким же ключом будет утеряна. (см. пример ниже).

```
public class Test {

    public Test() {
    }

    public static void main(String[] args) {
        Test t = new Test();
        TreeMap tm = new TreeMap();
        tm.put("key","String1");
        System.out.println(tm.get("key"));
        tm.put("key","String2");
        System.out.println(tm.get("key"));
    }
}
```

```
String1
String2
```

4.4. Класс Collections

Сразу же следует отметить, что не нужно путать класс java.util.Collections с интерфейсом java.util.Collection.

Класс Collections является классом-утилитой и содержит несколько вспомогательных методов для работы с классами обеспечивающими различные интерфейсы коллекций. Например для сортировки элементов списков, для поиска элементов в упорядоченных коллекциях и т.д. Но пожалуй наиболее важным свойством этого класса является возможность получения синхронизированный вариантов классов-коллекций. Например для получения синхронизированного варианта Map можно использовать следующий подход.

```
HashMap hm = new HashMap();
…
Map syncMap = Collections.synchronizedMap(hm);
…
```

Как уже отмечалось ранее, начиная с JDK 1.2 класс Vector реализует интерфейс List. Рассмотрим пример сортировки элементов содержащихся в классе Vector.

Следует еще раз напомнить, что в действительности в коллекции содержатся лишь ссылки на объекты, а не их копии.

```
public class Test {

    private class TestObject{
        private String name = "";
        public TestObject(String name){
            this.name = name;
        }
    }

    private class MyComparator implements Comparator{
        public int compare(Object l,Object r){
            String left = (String)l;
            String right = (String)r;
            return -1 * left.compareTo(right);
        }
    }

    public Test() {
    }

    public static void main(String[] args) {
        Test test = new Test();
        Vector v = new Vector();
```

```
        v.add("bbbbb");
        v.add("aaaaa");
        v.add("ccccc");
        System.out.println("Default elements order");
        test.dumpList(v);
        Collections.sort(v);
        System.out.println("Default sorting order");
        test.dumpList(v);
        System.out.println("Reverse sorting order with providing imlicit
comparator");
        Collections.sort(v,test.new MyComparator());
        test.dumpList(v);
    }

    private void dumpList(List l){
        Iterator it = l.iterator();
        while(it.hasNext()){
            System.out.println(it.next());
        }
    }
}
```

5. Класс Properties

Класс Properties предназначен для хранения набора свойств (параметров). Методы

```
String getProperty(String key)
String getProperty(String key,String defaultValue)
```

позволяют получить свойство из набора.

С помощью метода setProperty(String key, String value) это свойство можно установить.

Метод load(InputStream inStream) позволяет загрузить набор свойств из входного потока (Потоки данных подробно рассматриваются в главе 15). Как правило это текстовый файл в котором хранятся параметры. Параметры представляют собой строки представляющие сбой пары ключ/значение. Предполагается, что по умолчанию используется кодировка ISO 8859-1. Каждая строка должна оканчиваться символами \r,\n или \r\n. Строки из файла будут считываться пока не будет достигнут его конец. Строки состоящие из одних пробелов или начинающиеся со знаков ! или # игнорируются, т.е. их можно трактовать как комментарии. Если строка оканчивается символом /, то следующая строка считается ее продолжением. Первый символ с начала строки, отличающийся от пробела, считается началом ключа. Первый встретившийся пробел, :, = считается окончанием ключа. Все символы окончания ключа при необходимости могут быть включены в название ключа, но при этом перед ними должен стоять символ \. После того как встретился символ окончания ключа, все аналогичные символы будут проигнорированы до начала значения. Оставшаяся часть строки интерпретируется как значение. В строке, состоящей только из символов \t, \n, \r, \\, \", \', \ и \uxxxx, они все распознаются и интерпретируются как одиночные символы. Если встретится сочетание \ и символа конца строки, то следующая строка будет считаться

продолжением текущей, так же будут проигнорированы все пробелы до начала строки-продолжения.

Метод save(OutputStream inStream,String header) сохраняет набор свойств в выходной поток, в виде пригодном для вторичной загрузки с помощью метода load. Символы считающиеся служебными, кодируются так, что бы их можно было считать при вторичной загрузке. Символы в национальной кодировке будут приведены к виду \uxxxx . При сохранении используется кодировка ISO 8859-1. Если указан, header то он будет помещен в начало потока в виде комментария (т.е. с символом # в начале), далее будет следовать комментарий в котором будет указано время и дата сохранения свойств в потоке.

В классе Properties определено еще метод list(PrintWriter out) который практически идентичен save. Отличается лишь заголовок, который изменить нельзя. Кроме того строки усекаются по ширине. Поэтому этот метод для сохранения Properties не годится.

```
public class Test {

    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
        Properties props = new Properties();
        StringWriter sw = new StringWriter();
        sw.write("Key1 = Vlaue1 \n");
        sw.write("     Key2 : Vlaue2 \r\n");
        sw.write("     Key3  Vlaue3 \n ");
        InputStream is = new ByteArrayInputStream(sw.toString().getBytes());
        try {
            props.load(is);
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
        props.list(System.out);
        props.setProperty("Key1","Modified Value1");
        props.setProperty("Key4","Added Value4");
        props.list(System.out);
    }
}

-- listing properties --
Key3=Vlaue3
Key2=Vlaue2
Key1=Vlaue1
-- listing properties --
Key4=Added Value4
Key3=Vlaue3
Key2=Vlaue2
Key1=Modified Value1
```

6. Интерфейс Comparator

В коллекциях многие методы сортировки или сравнения требуют передачи в качестве одного из параметров объекта который реализует интерфейс Comparator. Этот интерфейс определяет единственный метод compare(Object obj1,Object obj2), который, на основании алгоритма определенного пользователем, сравнивает объекты переданные в качестве параметров. Метод compare должен вернуть

```
-1   если obj1 < obj2
0 если obj1 = obj2
1 если obj1 > obj2
```

7. Класс Arrays

Статический класс Arrays обеспечивает набор методов для выполнения операций над массивами, такие как поиск, сортировка, сравнение.В Arrays так же определен статический метод public List aList(a[] arr); который возвращает список фиксированного размера основанный на массиве. Изменения в List можно внести изменив данные в массиве.

Обратная операция, т.е. представление какой-либо коллекции в виде массива осуществляется с помощью статического метода Object[] toArray() определенного в классе Collections.

```
public class Test {
    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
        String[] arr = {"String 1","String 4","String 2","String 3"};
        test.dumpArray(arr);
        Arrays.sort(arr);
        test.dumpArray(arr);
        int ind = Arrays.binarySearch(arr,"String 4");
        System.out.println("\nIndex of \"String 4\" = " + ind);
    }
    void dumpArray(String arr[]){
        System.out.println();
        for(int cnt=0;cnt < arr.length;cnt++){
            System.out.println(arr[cnt]);
        }
    }
}
```

8. Класс StringTokenizer

Этот класс предназначен для разбора строки по лексемам (tokens). Строка которую необходимо разобрать передается в качестве параметра конструктору StringTokenizer(String str). Определено еще два перегруженных конструктора, которым дополнительно можно передать строку-разделитель лексем StringTokenizer(String str,String delim) и признак возврата разделителя лексем StringTokenizer(String str,String delim,Boolean retirnDelims)

Разделителем лексем по умолчанию служит пробел.

```
public class Test {

    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
        String toParse = "word1;word2;word3;word4";
        StringTokenizer st = new StringTokenizer(toParse,";");
        while(st.hasMoreTokens()){
            System.out.println(st.nextToken());
        }
    }
}

word1
word2
word3
word4
```

9. Класс BitSet

Класс BitSet предназначен для работы с последовательностями битов. Каждый компонент этой коллекции может принимать булево значение, которое обозначает установлен бит или нет. Содержимое BitSet может быть модифицировано содержимым другого BitSet с использованием операций AND, OR или XOR (исключающее или)

BitSet имеет текущий размер (количество установленных битов) может динамически изменятся. По умолчанию все биты в наборе устанавливаются в 0 (false). Установка и очистка битов в BitSet осуществляется методами set(int index) и clear(int index)

Метод int length() возвращает "логический" размер набора битов, int size() возвращает количество памяти занимаемой битовой последовательностью BitSet.

```
public class Test {

    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
```

```
        BitSet bs1 = new BitSet();
        BitSet bs2 = new BitSet();
        bs1.set(0);
        bs1.set(2);
        bs1.set(4);
        System.out.println("Length = " + bs1.length() + " size = " +
bs1.size());
        System.out.println(bs1);
        bs2.set(1);
        bs2.set(2);
        bs1.and(bs2);
        System.out.println(bs1);
    }
}
```

```
Length = 5 size = 64
{0, 2, 4}
{2}
```

Проанализировав первую строку вывода на консоль можно сделать вывод, что для внутреннего представления BitSet использует значения типа long.

10. Класс Random

Класс Random используется для получения последовательности псевдослучайных чисел. В качестве "зерна" используется 48 битовое число. Если для инициализации Random использовать одно и то же число, будет получена та же самая последовательность псевдослучайных чисел.

В классе Random определено так же несколько методов которые возвращают псевдослучайные величины для примитивных типов Java

Дополнительно следует отметить наличие двух методов double nextGaussian() - возвращает случайное число в диапазоне от 0.0 до 1.0 распределенное по нормальному закону, а void nextBytes(byte[] arr) - заполняет массив arr случайными величинами типа byte.

Пример использования Random

```
public class Test  {
    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
        Random r = new Random(100);
        // Generating the same sequence numbers
        for(int cnt=0;cnt<9;cnt++){
            System.out.print(r.nextInt() + " ");
```

```
        }
        System.out.println();
        r = new Random(100);
        for(int cnt=0;cnt<9;cnt++){
            System.out.print(r.nextInt() + " ");
        }
        System.out.println();
        // Generating sequence of bytes
        byte[] randArray = new byte[8];
        r.nextBytes(randArray);
        test.dumpArray(randArray);
    }

    void dumpArray(byte[] arr){
        for(int cnt=0;cnt< arr.length;cnt++){
            System.out.print(arr[cnt]);
        }
        System.out.println();
    }
}
```

```
-1193959466 -1139614796 837415749 -1220615319 -1429538713 118249332 -951589224

-1193959466 -1139614796 837415749 -1220615319 -1429538713 118249332 -951589224

81;-6;-107;77;118;17;93;-98;
```

11. Локализация

11.1. Класс Locale

Класс Locale предназначен для отображения определенного региона. Под регионом принято понимать не только географическое положение, но так же языковую и культурную среду. Так например для помимо того, что указывается страна Швейцария, можно указать так же и язык французский или немецкий.

Определено два варианта конструкторов в классе Locale

```
Locale(String language, String country)
Locale(String language, String country, String variant)
```

Первые два параметра в обоих конструкторах определяют язык и страну для которой определяется локаль, согласно кодировке ISO. (См приложения XXX1,XXX2). Список поддерживаемых стран и языков можно так же получить с помощью вызова статических методов Locale.getISOLanguages() Locale.getISOCountries() соответственно. Во втором варианте конструктора указан так же строковый параметр variant в котором кодируется информация о платформе. Если здесь необходимо указать дополнительные параметры,

то их необходимо разделить символом подчеркивания, причем более важный параметр должен следовать первым.

Пример использования

```
Locale l = new Locale("ru","RU");
Locale l = new Locale("en","US","WINDOWS");
```

Статический метод getDefault() возвращает текущую локаль, сконструированную на основе настроек операционной системы под управлением которой функционирует JVM.

Для наиболее часто использующихся локалей заданы константы. Например Locale.US или Locale.GERMAN.

После того как экземпляр класса Loale создан, с помощью различных методов можно получить дополнительную информацию о локали.

```
public class Test  {
    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
        Locale l = Locale.getDefault();
        System.out.println(l.getCountry() + " " + l.getDisplayCountry() + "
" + l.getISO3Country());
        System.out.println(l.getLanguage() + " " + l.getDisplayLanguage() +
" " + l.getISO3Language());
        System.out.println(l.getVariant() + " " + l.getDisplayVariant());
        l = new Locale("ru","RU","WINDOWS");
        System.out.println(l.getCountry() + " " + l.getDisplayCountry() + "
" + l.getISO3Country());
        System.out.println(l.getLanguage() + " " + l.getDisplayLanguage() +
" " + l.getISO3Language());
        System.out.println(l.getVariant() + " " + l.getDisplayVariant());
    }
}
```

```
US United States USA
en English eng


RU Russia RUS
ru Russian rus
WINDOWS WINDOWS
```

11.2. Класс ResourceBundle

Абстрактный Класс ResourceBundle предназначен для хранения объектов специфичных для локали. Например, когда необходимо получить набор строк, зависящих от локали используют ResourceBundle.

Использование ResourceBundle настоятельно рекомендуется, если предполагается использовать программу в многоязыковой среде. С помощью этого класса легко манипулировать, наборами ресурсов зависящих от локалей, легко их менять, добавлять новые и т.д.

Набор ресурсов - это фактически набор классов, имеющих одно базовое имя. Далее наименование класса дополняется наименованием локали, с которой связывается этот класс. Например, если имя базового класса будет MyResources, то для английской локали имя класса будет MyResources_en, для русской - MyResources_ru. Помимо этого может добавляться идентификатор языка, если для данного региона определено несколько языков. Например MyResources_de_CH так будет выглядеть швейцарский вариант немецкого языка. Кроме того можно указать дополнительную признак variant (см. описание Locale

). Так, описанный раннее пример для платформы UNIX будет выглядеть следующим образом: MyResources_de_CH_UNIX

Загрузка объекта для нужной локали производится с помощью статического метода getBundle.

```
ResourceBundle myResources =      ResourceBundle.getBundle("MyResources",
someLocale);
```

Если объект, имя которого в точности совпадает с именем, указанным в getBundle, то будет найден тот, имя которого максимально сходно с запрошенным. Приоритет перебора суффиксов будет такой. В начале ищется файл с указанной локалью, если такого файла нет, то будет подбираться класс с локалью по умолчанию. Если его тоже нет, то будет выбираться класс наименование которого совпадает с базовым классом.

```
baseclass + "_" + language1 + "_" + country1 + "_" + variant1
baseclass + "_" + language1 + "_" + country1 + "_" + variant1 + ".properties"

baseclass + "_" + language1 + "_" + country1
baseclass + "_" + language1 + "_" + country1 + ".properties"
baseclass + "_" + language1
baseclass + "_" + language1 + ".properties"
baseclass + "_" + language2 + "_" + country2 + "_" + variant2
baseclass + "_" + language2 + "_" + country2 + "_" + variant2 + ".properties"

baseclass + "_" + language2 + "_" + country2
baseclass + "_" + language2 + "_" + country2 + ".properties"
baseclass + "_" + language2
baseclass + "_" + language2 + ".properties"
baseclass
```

```
baseclass + ".properties"
```

Индексы 1 2 в данном случае подразумевают затребованную локаль и локаль по умолчанию.

Например если необходимо найти ResourceBundle для локали fr_CH (Швейцарский французский), а локаль по умолчанию en_US при этом название базового класса ResourceBundle MyResources, то порядок поиска подходящего ResourceBundle будет таков.

```
MyResources_fr_CH
MyResources_fr
MyResources_en_US
MyResources_en
MyResources
```

Результатом работы getBundle будет загрузка необходимого класса ресурсов в память, однако данные этого класса могут быть сохранены на диске. Т.о. если нужный класс не будет найден, то к требуемому имени класса будет добавлено расширение ".properties" и будет совершена попытка найти файл с данными на диске.

Следует помнить, что необходимо указывать полностью квалифицированное имя класса ресурсов т.е. имя пакета, имя класса. Кроме того, класс ресурсов должен быть доступен в контексте его вызова (т.е. там где вызывается getResourceBundle), те не быть private и т.д.

Всегда должен создаваться базовый класс без суффиксов, т.е. если вы создается ресурсы с именем MyResource, должен быть в наличии класс MyResource.class

ResourceBundle хранит объекты в виде пар ключ/значение. Как уже отмечалось ранее, класс ResourceBundle абстрактный, поэтому при его наследовании необходимо переопределить методы Enumeration getKeys()

```
public Object handleGetObject(String key)
```

первый метод должен возвращать список всех ключей, которые определены в Resource-Bundle, второй должен возвращать объект связанный с конкретным ключом.

Рассмотрим пример использования ResourceBundle.

```
public class MyResource extends ResourceBundle {

    private Hashtable res = null;
    public MyResource() {
        res = new Hashtable();
        res.put("TestKey","English Variant");
    }

    public Enumeration getKeys() {
        return res.keys();
    }

    protected Object handleGetObject(String key) throws
```

```
java.util.MissingResourceException {
        return res.get(key);
    }
}

public class MyResource_ru_RU extends ResourceBundle{
    private Hashtable res = null;

    public MyResource_ru_RU() {
        res = new Hashtable();
        res.put("TestKey","Русский варинат");
    }

    public Enumeration getKeys() {
        return res.keys();
    }

    protected Object handleGetObject(String key) throws
java.util.MissingResourceException {
        return res.get(key);
    }
}

public class Test  {
    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
        ResourceBundle rb =
ResourceBundle.getBundle("experiment.MyResource",Locale.getDefault());
        System.out.println(rb.getString("TestKey"));
        rb = ResourceBundle.getBundle("experiment.MyResource",new
Locale("ru","RU"));
        System.out.println(rb.getString("TestKey"));
    }
}


English Variant
Русский Вариант
```

Кроме того, следует обратить внимание, что ResourceBundle может хранить не только строковые значения. В нем можно хранить также двоичные данные или просто методы, реализующие нужную функциональность в зависимости от локали.

```
public interface Behavior {
  public String getBehavior();
  public String getCapital();
}
```

```
public class EnglishBehavior implements Behavior{
  public EnglishBehavior() {
  }
  public String getBehavior(){
     return "English behavior";
  }
  public String getCapital(){
    return "London";
  }
}

public class RussianBehavior implements Behavior {
  public RussianBehavior() {
  }
  public String getBehavior(){
    return "Русский вариант поведения";
  }
  public String getCapital(){
    return "Москва";
  }
}

public class MyResourceBundle_ru_RU extends ResourceBundle  {
  Hashtable bundle = null;
  public MyResourceBundle_ru_RU() {
    bundle = new Hashtable();
    bundle.put("Bundle description","Набор ресурсов для русской локали");
    bundle.put("Behavior",new RussianBehavior());
  }
  public Enumeration getKeys() {
    return bundle.keys();
  }
  protected Object handleGetObject(String key) throws
java.util.MissingResourceException {
    return bundle.get("key");
  }
}

public class MyResourceBundle_en_EN {
  Hashtable bundle = null;
  public MyResourceBundle_en_EN() {
    bundle = new Hashtable();
    bundle.put("Bundle description","English resource set");
    bundle.put("Behavior",new EnglishBehavior());
  }
  public Enumeration getKeys() {
    return bundle.keys();
  }
  protected Object handleGetObject(String key) throws
```

```
java.util.MissingResourceException {
    return bundle.get("key");
  }
}

public class MyResourceBundle extends ResourceBundle {
  Hashtable bundle = null;
  public MyResourceBundle() {
    bundle = new Hashtable();
    bundle.put("Bundle description","Default resource bundle");
    bundle.put("Behavior",new EnglishBehavior());
  }
  public Enumeration getKeys() {
    return bundle.keys();
  }
  protected Object handleGetObject(String key) throws
java.util.MissingResourceException {
    return bundle.get(key);
  }
}

public class Using {

  public Using() {
  }

  public static void main(String[] args) {
    Using u = new Using();
    ResourceBundle rb =
ResourceBundle.getBundle("lecture.MyResourceBundle",Locale.getDefault());
    System.out.println((String)rb.getObject("Bundle description"));
    rb = ResourceBundle.getBundle("lecture.MyResourceBundle",new
Locale("en","EN"));
    System.out.println((String)rb.getObject("Bundle description"));
    Behavior be = (Behavior)rb.getObject("Behavior");
    System.out.println(be.getBehavior());
    System.out.println(be.getCapital());
  }
}></eg>
    <eg><![CDATA[
Русский набор ресурсов
English  resource bundle
English behavior
London></eg>
    </div2>
    <div2 id="JAVA-LEC14-ListResourceBundle ">
     <head>Классы ListResourceBundle и PropertiesResourceBundle</head>
     <p>У класса <kw>ResourceBundle</kw> определено два прямых потомка
<kw>ListResourceBundle</kw> и <kw>PropertiesResourceBundle</kw>. </p>
```

```
     <p>
     <kw>PropertiesResourceBundle</kw>  хранит набор ресурсов в файле, который
 представляет собой  набор строк.  </p>
     <p>Следует акцентировать внимание, что в случае с
<kw>PropertiesResourceBundle</kw> данные хранятся в обычном текстовом файле
и соответственно можно иметь дело только со строковыми значениями. </p>
     <p>Алгоритм конструирования объекта содержащего набор ресурсов был описан
 в предыдущем параграфе. Во все случаях когда в качестве последнего элемента
 используется <code>.properties</code>, например  <code>baseclass + "_" +
language1 + "_" + country1 + ".properties"</code> речь идет о создании
<kw>ResourceBundle</kw> из файла с наименованием <code>baseclass + "_" +
language1 + "_" + country1</code> и расширением <code>properties. </code>Обычно
 класс <kw>ResourceBundle</kw> помещают в пакет <kw>resources</kw>, а файл
свойств в каталоге <kw>resources</kw>. Тогда для того что инстанциировать
нужный класс необходимо указать полный путь к этому классу(файлу) </p>
     <eg><![CDATA[
getBundle("resources.MyResource",Locale.getDefault());
```

ListResourceBundle хранит набор ресурсов в виде коллекции и является абстрактным классом. Классы которые наследуют ListResourceBundle должны обеспечить

- переопределение метода Object[][] getContents() который возвращает массив ресурсов.
- и собственно двумерный массив содержащий ресурсы.

Рассмотрим пример

```
public class MyResource extends ListResourceBundle {
    Vector v = new Vector();

    Object[][] resources = {
    {"StringKey","String"},
    {"DoubleKey",new Double(0.0)},
    {"VectorKey",v},
    };

    public MyResource(){
        super();
        v.add("Element 1");
        v.add("Element 2");
        v.add("Element 3");
    }

    protected Object[][] getContents() {
        return resources;
    }
}

public class Test  {
    public Test() {
```

```
    }
    public static void main(String[] args) {
        Test test = new Test();
        ResourceBundle rb =
ResourceBundle.getBundle("experiment.MyResource",Locale.getDefault());
        Vector v = (Vector)rb.getObject("VectorKey");
        Iterator it = v.iterator();
        while(it.hasNext()){
            System.out.println(it.next());
        }
    }
}

Element 1
Element 2
Element 3
```

Создание ресурсов для локалей отличных от локали по умолчанию, осуществляется так же, как это было показано для ResourceBundle.

Следует заострить внимание на том, что в отличие от PropertiesResourceBundle возможно задание ресурсов не только в виде строк, что требуется несомненно чаще всего, но и виде объектов. Например ничто не мешает сохранить в списке ресурсов скажем объект Image или звуковой трек.Следует избегать однако создания чересчур громоздких объектов содержащих ресурсы т.к. это увеличивает время их загрузки и расходует память.

Далее следует обратить внимание, что если определены как файл ресурсов так и объект с одинаковыми именами, то использован будет объект. Следует это из описания алгоритма поиска необходимого списка ресурсов. (см. выше)

Ключом в ResourceBundle может служить только строковое значение. Если в качестве ключа указать другой объект (это возможно в случае наследования ResourceBundle, ListResourceBundle), ошибки времени исполнения не возникнет, но данный ресурс будет недоступен. Т.к. в ResourceBundle определены методы получающие в качестве параметра объект типа String.

Например класс MyResource будет откомпилировано нормально.

```
public class MyResource extends ListResourceBundle {
    Vector v = new Vector();

    Object[][] resources = {
    {"Key1","String1"},
    {new Double(1.0),"Double value"}
    };


    protected Object[][] getContents() {
        return resources;
    }
}
```

Однако при попытке получить ресурс использовав в качестве параметра Double получим ошибку компиляции.

```
public class Test  {
    public Test() {
    }
    public static void main(String[] args) {
        Test test = new Test();
        ResourceBundle rb =
ResourceBundle.getBundle("experiment.MyResource",Locale.getDefault());
        System.out.println(rb.getString("Key1"));
        System.out.println(rb.getObject(new Double(1.0)));
    }
}
```

12. Заключение

- Для работы с датой и временем должны использоваться классы Date , Calendar. Класс Calendar абстрактный, существует конкретная реализация этого класса GregorianCalendar.
- Класс Observer и интерфейс Observable реализуют парадигму MVC и предназначены для уведомления одного объекта об изменении состояния другого.
- Коллекции (Collections) не накладывают ограничений на порядок следования и дублирование элементов.
- Списки (List) поддерживают порядок элементов. (управляется либо самими данными, либо внешними алгоритмами
- Наборы (Set) не допускают дублированных элементов.
- Карты (Maps) используют уникальные ключи, для поиска содержимого.
- Использование массивов делает добавление, удаление и увеличение количества элементов затруднительным.
- Использование связанных (LinkedList) облегчает вставку, удаление и увеличение размера хранилища, но снижает скорость индексированного доступа
- Использование деревьев (Tree) облегчает вставку, удаление и увеличение размера хранилища, снижает скорость индексированного доступа, но увеличивает скорость поиска.
- Использование хэширования облегчает вставку, удаление и увеличение размера хранилища, снижает скорость индексированного доступа, но увеличивает скорость поиска. Однако хэширование требует наличия уникальных ключей для для зпоминания элементов данных
- Класс Properties удобен для хранения наборов параметров в виде пар ключ/значение. Параметры могут сохраняться в потоки (файлы) и загружаться из них.

- Реализация классом интерфейса Comparator позволяет сравнивать экземпляры класса друг с другом и, соответственно, сортировать их, например в коллекциях.
- Arrays является классом-утилитой и обеспечивает набор методов, реализующих различные приемы работы с массивами. Не имеет конструктора.
- StringTokenizer - вспомогательный класс, предназначенный для разбора строк на лексемы.
- При необходимости работать с сущностями, представленными в виде битовых последовательностей, удобно использовать класс BitSet
- Манипуляцию ресурсов, различающихся в зависимости от локализации, удобно осуществлять с помощью классов ResourceBundle, ListResourceBundle, PropertiesResourceBundle. Собственно локаль задается с помощью класса Locale.

13. Контрольные вопросы

14-1. Необходимо написать метод, который возвращает случайное число в диапазоне от 0 до 100 кратное 5. Из перечисленных вариантов выберите правильный.

✓a.)
```
public int getRandom5(){
return (int)(Math.random()*20) * 5;
}
```

b.)
```
public int getRandom5(){
 Math m = new Math()
return (int)(m.random()*20) * 5;
}
```

c.)
```
public int getRandom5(){
return (Math.random()*20) * 5;
}
```

Правильный ответ a. Ответ b неверен т.к. производится попытка создать экземпляр класса Math, что вызовет ошибку компиляции т.к. Math не имеет конструктора. Ответ с не верен, потому что Math.random() возвращает случайную величину типа double в диапазоне от 0.0 до 1.0. Соответственно все выражение будет иметь тип double, а возвращаемое значение у функции типа int, следовательно, возникнет ошибка времени компиляции.

14-2. Какое из выражений относительно класса java.lang.Runtime является корректным?

✓a.) Объект Runtime создается при помощи следующего кода
```
Runtime r = Runtime.getRuntime();
```

b.) Метод gc() определенный в Runtime() вызывает начало сборки мусора виртуальной машиной Java

c.) Метод freeMemory() определенный в классе Runtime, освобождает не используемую память.

Правильный ответ a. Ответ b неверен т.к. процедура запуска сборки мусора запускается самой виртуальной машиной, gc() только уведомляет JVM о необходимости эту процедуру начать. Ответ c неверен, т.к. freeMemory возвращает количество свободной памяти, доступной JVM.

14-3. Необходимо написать метод, который получает в качестве параметра значение угла в градусах типа double и вычисляет его косинус. Какой из приведенных вариантов правилен.

a.)
```
double getCos(double angle){
    return Math.cos(angle);
}
```

✓b.)
```
double getCos(double angle){
    return Math.cos(angle * Math.PI / 180);
}
```

c.)
```
double getCos(double angle){
    return Math.cos(angle * PI / 180);
}
```

Правильный ответ b. Ответ a неверен т.к. тригонометрические функции получают в качестве параметра значение угла выраженное в радианах, а не в градусах. Ответ c неверен т.к. константа PI определена в классе Math.

14-4. В JDK 1.2 введены новые классы и интерфейсы, которые позволяют работать с наборами объектов. Отметьте те из них, которые являются интерфейсами.

✓a.) a java.util.List

b.) java.util.TreeMap

c.) java.util.AbstractList

✓d.) java.util.SortedMap

✓e.) java.util.Iterator

f.) java.util.Collections

Правильные ответы a, d, e. TreeMap является конкретным классом, AbstractList абстрактный класс, Collections класс-утилита.

14-5. Какие высказывания относительно java.util.Vector и java.util.Hashtable можно считать корректными

a.) В Vector могут сохранятся ссылки как на объекты так и на примитивные типы.

✓b.) Ссылки на объекты в Vector хранятся в порядке их добавления.

c.) В качестве ключей для Hashtable должны передаваться оъекты типа String

d.) Ссылки на объекты в Hashtable хранятся в порядке их добавления.

✓e.) И Hashtable и Vector являются синхронизированными, для того что бы избежать проблема, когда несколько потоков пытаются получить доступ к одной и той же коллекции.

Правильные ответы b, e. Ответ a неверен т.к. в классе Vector могут храниться лишь ссылки на объекты. Для манипулирования примитивными типами используются соответствующие классы-обертки. Ответ c неверен, т.к. в качестве ключа, в Hashtable может использоваться любой объект не только строки. Ответ d неверен, т.к. порядок следования объектов в Hashtable непредсказуем.

14-6. Приведенный ниже пример кода вызывает ошибку компиляции.

```
    double getCos(double angle){
        return Math.cos(angle);
    }

public static void showStatus(Boolean flag){
 if(flag){
  System.out.println("FIRED")
 }else{
  System.out.println("NOT READY");
 }
}
```

Какое из перечисленных ниже исправлений решит проблему?

a.) Заменить if(flag){ на if(flag.equals(true)

✓b.) Заменить public static void showStatus(Boolean flag){ на public static void showStatus(boolean flag){

✓c.) Заменить if(flag){ на if(flag.booleanValue()){

Правильные ответы b, c. Ответ a неверен т.к. equals метод требует в качестве параметра ссылку на объект, а true есть примитивный тип Boolean.

14-7. Какое значение будет выведено на консоль в существующем фрагменте кода?

```
String str1 = "abc";
String str2 = "abc";
System.out.println(str1 == str2);
```

✓a.) true

a.) false

Правильный ответ a. Т.к. строковые литералы, которыми инициализируются строки str1 и str2 одинаковы, то ссылаться они будут на один и тот же объект.

14-8. Будет ли переменная sb после выполнения кода в строке 2 указывать на тот же самый объект?

```
1. StringBuffer sb = new StringBuffer("abc");
2. sb.append("x");
```

✓a.) Да

b.) Нет

Правильный ответ a. Новый объект создается лишь при манипуляциях с объектом класса String.

14-9. Какой из перечисленных ниже классов имеет набольшее сходство с классом Vector?

a.) TreeSet.

b.) AbstractCollection.

✓c.) c ArrayList

d.) d Hashtable

Правильный ответ c т.к. именно ArrayList наиболее схож по функциональности с Vector. Ответ a неверен т.к. TreeSet использует сортировку. Ответ b неверен, т.к. AbstractCollection абстрактный класс. Ответ d неверен, т.к. Hashtable использует для хранения пары ключ/значение.

14-10. Какой из перечисленный ниже интерфейсов реализует Hashtable?

a.) a SortedMap

✓b.) b Map

c.) c List

d.) d SortedSet

e.) e ни один из перечисленных

Правильный ответ b. Ответы a, c, d не верны, т.к. Hashtable не реализует эти интерфейсы. Ответ e не верен, т.к. в JDK 1.2 Hashtable переделан и реализует интерфейс Map.