此算法是我个人研究的，经过测试证明我的算法还是不错的。

```
一共进行了100000次抽样,每次从1-9这9个数中抽取3个数
每个数被抽中次数的参考值为(3/9)*100000约33333次
每个数被抽到的次数依次为:33124,33399,33352,33085,33395,33479,33246,33300,33620
本次测试用时:0.0879926918秒
```

PS：这里的时间可能有点偏小，实际用时是2秒左右，我没有去研究原因了。

算法实现的功能

从一个群体(大小为N的数组)中随机抽取一定数量(M个)的样本
即
从一个大小为N的int数组中随机抽取M个不重复的元素放到一个新数组中

算法的设计思想

首先需要准备要被抽样的数组num1和存放抽样结果的数组num2
然后在M次循环中每次随机抽取一个数存入num2中
如果，每次从1到N这N个数中随机抽取一个整数作为被抽取的位置的话，那么可能会抽取到重复的数字，因此我这里需要产生的随机数应该是原数组去掉已经被抽取的位置之后的位置，你可能会想到每次抽取一个数，就将这个数从原数组里面去掉，然后再用剩下的元素重新组成数组，但是这样的话效率会很低，我的想法是用一个list保存每次抽取的位置，由于每次抽取之后剩余的可抽取的数量都会减1，所以产生随机数的范围应当是N减去已经抽取的次数，这样产生的随机i数代表了第i个剩余的元素，比如从1-10这10个数中抽取5个数，那么第3次抽取的时候，应当从1-8这8个数中随机选一个数，代表剩下的8个数中的第几个数，然后通过一个方法处理一下，就可以得出这个第几个在原来的数组中实际对应的位置，这部分是这个算法的关键之处。

算法的具体实现及准确率、效率测试

```
/**
 * 从一个集合中随机抽取一定数量的不重复的样本
 */
package 算法设计;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class RandomUtil {

 /**
  * @param args
  */
 public static void main(String[] args) {
   //test();
    testChance(9,3,100000);
 }

 //num:要抽取样本的集合
 //k:需要抽取样本的数量
 public static int[] getSimple(int[] num,int k){
   if(num==null||num.length==0||num.length<k){
     return new int[0];
   }
```

```
    int[] simple = new int[k];//存放样本
    int size = num.length;//集合的大小
    List disabled = new ArrayList();//存放已被抽中的元素的位置

    for(int i=0;i<k;i++){//k次循环,每次抽取一个样本
      Random r = new Random();
      // 每抽取一个样本后，集合中剩下的可抽取的元素数量减少1个
      // 如从10个元素中抽取样本，抽第四个时，产生的随机数应该从1-7中选出，分表代表剩下的7个
      int where = r.nextInt(size-i);//nextInt(n)返回0到n-1的随机值
      //System.out.println(where);
      //pos为在num中的实际位置，指的是第where个未被抽取的元素的位置
      int pos = get(where,disabled);
      simple[i] = num[pos];
      disabled.add(new Integer(pos));//将已被抽取的位置保存起来
    }

    return simple;
  }

  /**
   * 根据已经抽取过的位置,来确定剩下的第i个元素的位置
   * 由于选取随机数的时候已经限定了i的最大值不会超过剩下元素的个数，所以不用担心数组越界
   * @param i 第i个位置
   * @param disabled 已经抽中的列表
   * @return 第i个位置排除了已经抽中元素后在num数组中实际对应的位置
   */
  public static int get(int i,List disabled){
    int pos = i;
    Collections.sort(disabled);//排序
    for(int k=0;k<disabled.size();k++){
      //如果在第i个元素之前有已经被抽中的元素，则i应该是后一个元素即i+1
      if(((int)(Integer)disabled.get(k))<=pos){
        pos++;
      }
    }
    return pos;
  }

  //打印一个int数组
  public static void printArr(int[] num){
    for(int i=0;i<num.length-1;i++){
      System.out.print(num[i]+",");
    }
    System.out.println(num[num.length-1]);
  }

  //测试本类中的get方法
  public static void test(){
    List list = new ArrayList();
    list.add(new Integer(2));
    list.add(new Integer(6));
    list.add(new Integer(1));
```

```
    list.add(new Integer(3));
    list.add(new Integer(5));
    Collections.sort(list);
    System.out.println(list.toString());
    System.out.println(get(3,list));
  }

  //得到一个从1开始的大小为k的数组
  public static int[] getIntArr(int k){
    int[] num = new int[k];
    for(int i=0;i<num.length;i++){
      num[i] = i+1;
    }
    return num;
  }

  /**
   * 测试随机方法对每一位数的概率
   * @param size 被抽样集合的大小
   * @param simpleSize 每次抽样的数量
   * @param times 随机抽样的次数
   */
  public static void testChance(int size,int simpleSize,int times){
    //计时开始
    long startTime = System.nanoTime();

    int[] count = new int[size];//统计每一个元素的出现总数
    int[] num = getIntArr(size);
    int[] num2;
    for(int i=0;i<times;i++){
      num2 = getSimple(num,simpleSize);
      count(num2,count);
    }
    System.out.println("一共进行了"+times+"次抽样,每次从1-"+size+"这"+size+"个数
中抽取"+simpleSize+"个数");
    System.out.println("每个数被抽中次数的参考值为
("+simpleSize+"/"+size+")*"+times+"约"+(int)(simpleSize*times/size)+"次");
    System.out.print("每个数被抽到的次数依次为:");
    printArr(count);

    //计时结束
    long endTime = System.nanoTime();
    System.out.println("本次测试用时:"+(endTime-startTime)/10e9+"秒");
  }

  /**
   * @param num 需要被统计的数组
   * @param count 存放统计数据的数组
   */
  public static void count(int[] num,int[] count){
    for(int i=0;i<num.length;i++){
      count[num[i]-1]++;
    }
  }
}
```