```
package bingoballs;

import java.util.*;

/**
 *
 * @author agomezdg
 */
public interface SetADT<T> extends Iterable<T> {
    /* Adds one element to this set, avoiding duplication: */
    public void add(T element);

    /* Returns true if this set contains the parameter: */
    public boolean contains(T target);

    /* Removes and returns a random element from this set: */
    public T removeRandom();

    /* Removes and returns the specified element from this set: */
    public T remove(T element);

    /* Returns true if this set contains no elements: */
    public boolean isEmpty();

    /* Returns the number of elements in this set: */
    public int size();

    /* Adds all the elements of the parameter to this set (merges
     * another set with this one): */
    public void addAll(SetADT<T> set);

    /* Returns true if this set and the parameter contain exactly
     * the same elements: */
    public boolean equals(SetADT<T> set);

    /* Returns an iterator over this set: */
    public Iterator<T> iterator();

    /* Prints the contents of this set: */
    public String toString();
}


import java.util.*;

/**
 *
 * @author agomezdg
 */
public class ArrayIterator<T> implements Iterator<T> {
    private int count;    // the number of elements in the collection
    private int current;  // the current position in the iteration
    private T[] items;

    public ArrayIterator(T[] collection,int size) {
        items=collection;
        count=size;
        current=0;
    }

    public boolean hasNext() {
        return current<count;
    }

    public T next() {
        if(!hasNext())
            throw new NoSuchElementException();
        else{
            current++;
            return items[current-1];
        }
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}


import java.util.*;
/**
 *
 * @author agomezdg
 */
public class ArraySet<T> implements SetADT<T> {
    private static final int DEFAULT_CAPACITY=100;
    private static Random rand=new Random();
    private static final int NOT_FOUND=-1;
    private int count;  // the current number of elements in the set
    private T[] contents;

    public ArraySet(int initialCapacity) {
        count=0;
        contents=(T[])(new Object[initialCapacity]);
    }

    public ArraySet() {
        this(DEFAULT_CAPACITY);
    }

    public void add(T element) {
        if(!(contains(element))) {
            if(size()==contents.length)
                expandCapacity();
            contents[count]=element;
            count++;
        }
    }

    private void expandCapacity() {
        T[] larger=(T[])(new Object[contents.length*2]);
        for(int index=0;index<contents.length;index++)
            larger[index]=contents[index];
        contents=larger;
    }

    public boolean contains(T target) {
        return searchTarget(target)!=NOT_FOUND;
    }

    private int searchTarget(T target) {
        int search=NOT_FOUND;
        int index=0;
        while(index<count&&search==NOT_FOUND) {
            if(contents[index].equals(target))
                search=index;
            index++;
        }
        return search;
    }

    public T removeRandom() {
        if(isEmpty())
            throw new EmptyCollectionException();
        else {
            int choice=rand.nextInt(count);
            T result=contents[choice];
            contents[choice]=contents[count-1];  // fill the gap
            contents[count-1]=null;
            count--;
            return result;
        }
    }

    public T remove(T target) {
        if(isEmpty())
            throw new EmptyCollectionException();
        else {
            int search=searchTarget(target);
            if(search==NOT_FOUND)
                throw new NoSuchElementException();
            else {
                T result=contents[search];
                contents[search]=contents[count-1];
                contents[count-1]=null;
                count--;
                return result;
            }
        }
    }

    public boolean isEmpty() {
        return size()==0;
    }

    public int size() {
        return count;
    }

    public void addAll(SetADT<T> set) {
        for(T element:set)
            add(element);
    }

    public boolean equals(SetADT<T> set) {
        boolean result=false;
        int countEquals=0;

        if(size()==set.size()) {
            for(T element:set)
                if(contains(element))
                    countEquals++;
            result=countEquals==size();
        }

        return result;
    }

    // Podría haberse escrito de una manera un poco más eficiente así:
/*    public boolean equals(SetADT<T> set) {
        boolean result = false;
        Iterator<T> it=set.iterator();

        if(size()==set.size()) {
            while(it.hasNext()&&contains(it.next()));
            if(!it.hasNext())
                result=true;
        }

        return result;
    }*/

    public Iterator<T> iterator() {
        return new ArrayIterator<T>(contents, count);
    }

    public String toString() {
      StringBuilder result=new StringBuilder();
      for(int index=0;index<count;index++)
            result.append(contents[index].toString()+"\n");
      return result.toString();
    }
}


public class EmptyCollectionException extends RuntimeException{
    public EmptyCollectionException() {
        super("The collection is empty.");
    }

    public EmptyCollectionException(String message) {
        super(message);
    }
}


public class BingoBall {
```

```
private int number;
private char letter;

public BingoBall(int num) {
    this.number=num;
    if(num<=15)
        letter='B';
    else if(num<=30)
        letter='I';
    else if(num<=45)
        letter='N';
    else if(num<=60)
        letter='G';
    else
        letter='O';
}

public char getLetter() {
    return letter;
}

public int getNumber() {
    return number;
}

@Override
public boolean equals(Object obj) {
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final BingoBall other = (BingoBall) obj;
    if (this.number != other.number) {
        return false;
    }
    if (this.letter != other.letter) {
        return false;
    }
    return true;
}

@Override
public int hashCode() {
    int hash = 5;
    hash = 97 * hash + this.number;
    hash = 97 * hash + this.letter;
    return hash;
}

@Override
public String toString(){
    StringBuilder sb=new StringBuilder("");

    sb.append(letter+" "+number);
```

```
        return sb.toString();
    }
}


import java.util.*;

/**
 *
 * @author agomezdg
 */
public class Main {
    public static void main(String args[]){
        final int NUM_BALLS=75;
        final int NUM_PULLS=10;
        ArraySet<BingoBall> bingoSet=new ArraySet<BingoBall>();
        BingoBall ball;

        for(int num=1;num<=NUM_BALLS;num++) {
            ball=new BingoBall(num);
            bingoSet.add(ball);
        }

        System.out.println("Conjunto contiene 1: "+bingoSet.contains(new
BingoBall(1)));
        System.out.println("Tamaño del conjunto: "+bingoSet.size()+"\n");

        System.out.println("Bolas seleccionadas y removidas:");
        for(int num=1;num<=NUM_PULLS;num++) {
            ball=bingoSet.removeRandom();
            System.out.println(ball);
        }

        // Se puede usar el toString de la clase ArraySet para ver qué
        // contiene el conjunto al final:
        System.out.println("\nBolas restantes usando toString:");
        System.out.println(bingoSet.toString());

        // Se podría usar un 'for' tradicional para "visitar" cada objeto
        // del conjunto y luego verlo (usando el toString de la clase
        // BingoBall)...pero esto termina eliminando los objetos del
        // conjunto, ya que tenemos un método removeRandom pero no
tenemos
        // nada equivalente a peek, así es que hay que restaurarlo al
final:
        System.out.println("\nBolas restantes usando 'for'
tradicional:");
        BingoBall pelotita;
        ArraySet<BingoBall> aux=new ArraySet<BingoBall>();
        int j=bingoSet.size();
        for(int i=0;i<j;i++) {
            pelotita=bingoSet.removeRandom();
            System.out.println(pelotita);
            aux.add(pelotita);
        }
        bingoSet=aux;
        // Se puede usar esta otra forma del 'for' para visitar los
        // diversos elementos del conjunto e imprimirlos sin eliminarlos:
        System.out.println("\nBolas restantes usando 'for' nuevo:");
        for(BingoBall pelota:bingoSet)
            System.out.println(pelota);

        // O se puede usar un iterador para visitar los diversos
elementos
        // del conjunto e imprimirlos sin eliminarlos:
        System.out.println("\nBolas restantes usando iterador");
        Iterator<BingoBall> it = bingoSet.iterator();
        while(it.hasNext()) {
            ball=it.next();
            System.out.println(ball);
        }
    }
}
```