```java
public class AboveLine extends ScoreEntry
{
    private int myCount;

    public AboveLine(int count)
    {
        myCount = count;
    }

    public int calcScore(DiceGroup dg)
    {
        return 0;
    }

    public String getDescription()
    {
        return myCount + ": (score # " + myCount + "'s x " + myCount + ")";
    }
}
```



```java
public class Chance extends ScoreEntry
{
    public Chance()
    {

    }

    public String getDescription()
    {
        return "Chance (all)";
    }

    public int calcScore(DiceGroup dg)
    {
        return 0;
    }
}
```

```java
/**
 * This class represents a group/cup of dice that can
 * be rolled as a group, then some of the dice in the cup/group
 * can be selected as frozen and the others re-rolled
 *
 * When a CupOfDice is constructed the number of dice in
 * the cup/group are specified
 *
 * The dice in a cup/group are indexed from 0 to size()-1 so
 * in a cup of three dice legal indices are 0, 1, and 2
 *
 * An individual dice can be frozen (won't be rolled when
 * all dice in group rolled) using freeze(..)
 *
 * All dice can be unfrozen at once using unfreezeAll()
 *
 * Example:
 * TextDiceViewer view = new TextDiceViewer();
 * CupOfDice dg = new CupOfDice(5);            // create group of 5 dice
 * dg.roll();          // roll all dice, viewer shows group
 * dg.freeze(0);       // freeze dice #0
 * dg.roll();          // viewer shows dice #0 same as first time
 * dg.unfreezeAll();   // dice #0 now free (no dice frozen)
 * dg.roll()           // all dice could be different from first roll
 *
 * @author Owen Astrachan
 */

import java.util.Arrays;

public class CupOfDice extends DiceGroup
{
    /**
     * The parallel vectors/arrays myFrozen and myValues
     * determine how the dice are rolled.  If myFrozen[k] == false
     * then dice #k is *NOT* frozen, i.e., it's free to roll.  If
     * myFrozen[k] == true, then dice #k is frozen
     * and it won't be rolled when the dicegroup is rolled
     *
     * See code in roll() for how these vectors work
     * together
     *
     */
    private int[] myValues;
    private boolean[] myFrozen;
    private Dice myDie;
    private int mySize;

    CupOfDice(int size)
    {
        myValues = new int[size];
        Arrays.fill(myValues,0);

        myFrozen = new boolean[size];
        Arrays.fill(myFrozen,false);

        myDie = new Dice(6);
        mySize = size;
    }

    /**
     * All unfrozen dice are rolled
     */

    public void roll()
    {
        for(int k = 0; k < mySize; k++) {
            if (!myFrozen[k]) {
                myValues[k] = myDie.roll();
            }
        }
        setChanged();
        notifyObservers();
```



```java
    }

    /**
     * Freeze die with specified index, 0 <= dIndex < size()
     * @param dIndex is the index of the die to be frozen
     */
    public void freeze(int dIndex)
    {
        myFrozen[dIndex] = true;
    }

    /**
     * All dice in this group are unfrozen
     */

    public void unfreezeAll()
    {
        for(int k = 0; k < mySize; k++) {
            myFrozen[k] = false;
        }
    }

    /**
     * Returns number of dice in this group.
     * @return number of dice in group
     */

    public int size()
    {
        return mySize;
    }

    /**
     * Return value of specific die in group
     * @param dIndex is index of die returned,  0 <= dIndex < size()
     * @return value shown on dIndex die in group
     */

    public int getValue(int dIndex)
    {
        return myValues[dIndex];
    }

    public static void main(String[] args)
    {
        CupOfDice d5 = new CupOfDice(5);
        for(int k = 0;k < 5; k++){
            d5.roll();
        }
    }
}
```

```
import java.util.Random;

/**
 * Class for simulating a die
 * (an object "rolled" to generate a random number)
 *
 */
public class Dice
{
    private int myRollCount;   // # times rolled
    private int mySides;       // # sides
    private static Random ourRandGen = new Random();  // the random number gener
ator


    /**
     * Construct standard six-sided Dice object
     */

    public Dice()
    {
        this(6);
    }

    /**
     * Constructs a Dice object.
     * @param sides specifies the number of "sides" for the die, e.g.,
     * 2 is a coin, 6 is a "regular" die.
     */
    public Dice(int sides)
    {
        myRollCount = 0;
        mySides = sides;
    }

    /**
     * Returns the random "roll" of the die, a uniformly distributed
     * random number in the range [1..numSides()].
     * @return a random value in range [1..numSides()]
     */
    public int roll()
    {
        myRollCount++;
        return ourRandGen.nextInt(mySides) + 1;
    }

    /*
     * Returns number of sides of die
     * @return number of sides of this dice object
     */

    public int numSides()
    {
        return mySides;
    }

    /*
     * Returns number of times this die object has been rolled.
     * @return number of times dice object rolled
     */

    public int numRolls()
    {
        return myRollCount;
    }

    /**
     * Returns a string representing this Dice object
     */

    public String toString()
    {
        return "# sides: " + numSides() + " # rolls: " + numRolls();
```



```
    }

    public static void main(String[] args)
    {
        Dice die = new Dice(6);

        for(int k = 0; k < 100; k++)
        {
            int roll = die.roll();
            System.out.println("roll " + die.numRolls() + ": " + roll);
        }
    }
}
```

```java
import java.util.Observable;
/**
 * This class represents a group of dice whose
 * individual die-values can be obtained. This is
 * an interface inherited, e.g., by CupOfDice for
 * playing a game like Yahtzee and by FixedDice for
 * testing purposes
 *
 * Access functions provide size of group and getting
 * values of each Dice object in the group
 *
 * @author Owen Astrachan
 */
public abstract class DiceGroup extends Observable
{
    /**
     * Returns number of dice in this group.
     * @return number of dice in group
     */
    public abstract int size();

    /**
     * Return value of specific die in group
     * @param dIndex is index of die returned,  0 <= dIndex < size()
     * @return value shown on dIndex die in group
     */
    public abstract int getValue(int dIndex);
}
```



```java
public class FiveKind extends ScoreEntry
{
    public FiveKind()
    {

    }

    public String getDescription()
    {
        return "YAHTZEE (50)";
    }

    public int calcScore(DiceGroup dg)
    {
        return 0;
    }
}
```

```java
/**
 * This class represents a DiceGroup but the values
 * in the group are fixed via construction or
 * calling method set.  The values are specified by
 * a string of digits 1-9, so only Dice of 9 or fewer
 * sides can be represented.
 *
 * FixedDice fd("23456");
 *
 * makes fd represent a group of five dice, with values
 * 2 3 4 5 6, so that fd.getValue(0) == 2, fd.getValue(1) == 3
 * fd.getValue(2) == 4, fd.getValue(3) == 5, and fd.getValue(4) == 6
 *
 * Values can be set also using method set(String) so that
 * fd.set("1111") will make fd a group of four dice, all with value 1
 *
 * @author Owen Astrachan
 *
 */
public class FixedDice extends DiceGroup
{
    private int[] myValues;

    /**
     * Construct a DiceGroup based on values in pips (all values
     * in pips should be digits '0'--'9', but non-digit characters
     * are ignored.)
     * @param pips consists of digits
     */
    public FixedDice(String pips)
    {
        myValues = new int[pips.length()];
        set(pips);
    }

    /**
     * Set this group to represent values stored in pips (all values
     * in pips should be digits '0'--'9', but non-digit characters
     * are ignored.)
     */
    public void set(String pips)
    {
        if (pips.length() > myValues.length) {
            myValues = new int[pips.length()];
        }
        int dIndex = 0;
        for(int k = 0; k < pips.length(); k++){
            if(Character.isDigit(pips.charAt(k))){
                        myValues[dIndex] = pips.charAt(k)-'0';
                        dIndex++;
            }
        }

        // check if values ignored, resize myValues if so
        if (dIndex != myValues.length) {
            int[] temp = new int[dIndex];
            System.arraycopy(myValues,0,temp,0,dIndex);
            myValues = temp;
        }
        setChanged();
        notifyObservers();
    }

    /**
     * Returns number of dice in this group.
     * @return number of dice in this group
     */

    public int size()
    {
        return myValues.length;
    }
```



```java
    /*
     * Returns value of specified die, 0 <= dIndex < size()
     * @return value of specified die
     */

    public int getValue(int dIndex)
    {
        return myValues[dIndex];
    }
}
```

```java
public class FourKind extends ScoreEntry
{
    FourKind()
    {

    }

    public String getDescription()
    {
        return "4 of a kind (all)";
    }

    public int calcScore(DiceGroup dg)
    {
        return 0;
    }

}
```



```java
public class FullHouse extends ScoreEntry
{
    FullHouse()
    {

    }


    public String getDescription()
    {
        return "Full House (25)";
    }

    public int calcScore(DiceGroup dg)
    {
        return 0;
    }
}
```

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

/**
 * This class rolls a DiceGroup object
 * a set # times with possible user-chosen
 * freezing happening in between each roll of the group
 * for selected dice
 *
 * e.g., roll "2 3 4 5"
 * user could freeze the last die and then roll the
 * first three again
 *
 * The function roll takes a DiceGroup as a parameter
 * and rolls the group # times specified by howMany
 *
 */
public class GroupRoller
{
    public void GroupRoller()
    {
    }

    public void roll(CupOfDice dg, int howMany)
    {
        int k;
        String line = "";
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));
        int index;

        dg.unfreezeAll();
        dg.roll();

        System.out.println("Pressing return freezes NO dice.");
        System.out.println();

        for(k=0; k < howMany-1; k++) {

            System.out.print("freeze which dice [e.g., \"034\"] ? ");

            try{
                line = in.readLine();
            }
            catch(IOException e) {
                e.printStackTrace();
                System.exit(0);
            }
            // process all chars on the line just read

            for(index = 0; index < line.length(); index++) {
                if (Character.isDigit(line.charAt(index))) {
                    int die = line.charAt(index) - '0';
                    if (0 <= die && die < dg.size()) {
                        dg.freeze(die);
                    }
                }
            }
            dg.roll();
            dg.unfreezeAll();
        }
    }

    public static void main(String[] args)
    {
        CupOfDice d5 = new CupOfDice(5);
        GroupRoller gr = new GroupRoller();
        TextDiceViewer viewer = new TextDiceViewer();
        d5.addObserver(viewer);

        gr.roll(d5,3);
    }
```



```
}
```

```java
public class LargeStraight extends ScoreEntry
{
    LargeStraight()
    {

    }


    public String getDescription()
    {
        return "Large Straight (40)";
    }

    public int calcScore(DiceGroup dg){
        return 0;
    }
}
```



```java
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.ArrayList;

/**
 * A Yahtzee scorecard (model)
 * A DiceGroup is passed to the score function and
 * the user prompted for how to score the DiceGroup
 *
 * the function allFilled() returns true when every
 * scorecard entry has a value (has been chosen by user)
 *
 * @author Owen Astrachan
 */
public class ScoreCard
{
    private static final int UPPER_BONUS = 35;
    private static final int INVOKE_BONUS = 63;
    private ArrayList myEntries;
    private int myCount;

    /**
     * Constructs a new ScoreCard object such that each entry is
     * is not scored
     */
    public ScoreCard()
    {
        myEntries = new ArrayList();
        init();
    }

    /**
     * Returns above line score.
     */
    public int upperScore()
    {
        int score = 0;
        for (int k=0; k < 6; k++){
            int s = ((ScoreEntry)myEntries.get(k)).getScore();
            if (s < 0) {
                s = 0;
            }
            score += s;
        }
        return score;
    }

    /**
     * Returns upper bonus (0 or bonus).
     * @return the upper bonus.
     */
    public int upperBonus()
    {
        int bonus = 0;
        if (upperScore() >= INVOKE_BONUS){
            bonus = UPPER_BONUS;
        }
        return bonus;
    }

    /**
     * Returns the score below the line.
     * @return below the line score
     */

    public int lowerScore()
    {
        int score = 0;
        for (int k=6; k < myEntries.size(); k++){
            int s = ((ScoreEntry)myEntries.get(k)).getScore();
            if (s < 0) {
                s = 0;
```

```java
            }
            score += s;
        }
        return score;
    }

    /**
     * Returns the total score of this ScoreCard object.
     * @return the total score for this scorecard
     */
    public int totalScore()
    {
        return upperScore() + upperBonus() + lowerScore();
    }

    /**
     * Returns true if card is full (no entry to score)
     * @return true iff there are no entries remaining to score
     */

    public boolean allFilled()
    {
        return myCount == 0;
    }

    /**
     * Clears the ScoreCard and fills it with new entries.
     */

    public void clear()
    {
        myEntries.clear();
        init();
    }

    public ScoreEntry[] getEntries()
    {
        return (ScoreEntry[])myEntries.toArray(new ScoreEntry[0]);
    }

    /**
     * Fills the ScoreCard with new entries.
     */
    private void init()
    {
        for(int k = 1; k <= 6; k++){
            myEntries.add(new AboveLine(k));
        }
        myEntries.add(new ThreeKind());
        myEntries.add(new FourKind());

        myEntries.add(new FullHouse());
        myEntries.add(new SmallStraight());
        myEntries.add(new LargeStraight());

        myEntries.add(new FiveKind());
        myEntries.add(new Chance());

        myCount = myEntries.size();
    }
}
```



```java
/**
 * A ScoreEntry is one of the entries on a Yahtzee score card
 * such as ones, twos, full house, three of a kind, etc.
 *
 * A ScoreEntry can score itself using a dicegroup since the
 * entry knows what kind it is (e.g., full house); it can
 * report its score and show a description of itself
 *
 * This is an interface/abstract class, subclasses
 * would include FullHouse, SmallStraight, FourKind, etc.
 * including some class for Ones, Twos, ... above the line entries
 *
 * @author Owen Astrachan
 *
 * int calcScore(dg) -- returns score based on dg, this doesn't
 *                      score the entry, but tells what the score
 *                      would be, SUBCLASSES MUST IMPLEMENT
 *
 * String getDescription() -- returns a string to display/show user
 *                            including score for entry,
 *                            SUBCLASSES MUST IMPLEMENT
 *
 *
 * --
 * The methods below are the same for all subclasses since
 * scoreIt just calls calcScore, and getScore returns the result
 * obtained by calling scoreIt, and -1 by default
 *
 * int getScore() -- returns current score, returns -1 if not scored yet
 *
 * void scoreIt(dg) -- score this entry (write down score on
 *                     scorecard), based on dg, getScore() will
 *                     then return the score based on dg
 *
 */
public abstract class ScoreEntry
{
    protected int myScore;

    /**
     * Constructs a new ScoreEntry object
     */

    public ScoreEntry()
    {
        myScore = -1;
    }

    /**
     * Returns score for this entry based on dg.
     * Doesn't score the ScoreEntry object, but tells
     * what the score would be
     */

    public abstract int calcScore(DiceGroup dg);

    /**
     * Returns a String to display/show user including the score for
     * this ScoreEntry.
     */

     public abstract String getDescription();

    /**
     * Returns the score of this ScoreEntry object.
     * @return the score for this ScoreEntry object
     */

    public int getScore()
    {
        return myScore;
    }
```

```java
    /**
     * Record the score for dg: sets state so that getScore()
     * return calcScore(dg), i.e., this ScoreEntry object
     * has been scored forever based on dg
     */

    public void scoreIt(DiceGroup dg)
    {
        myScore = calcScore(dg);
    }

    /**
     * Returns how many n's occur in dg. For DiceGroup dg = [1,2,1,3,1],
     * countN(dg,1) == 3, countN(dg,2) == 1, countN(dg,3) == 1, and
     * countN(dg,4) == 0.
     * @param dg is the group being counted,
     * @param n is the value in the group being counted
     * @return the number of occurrences of n in dg
     */
    protected static int countN(DiceGroup dg, int n)
    {
        int count = 0;
        for(int k=0; k < dg.size(); k++) {
            if (dg.getValue(k) == n) {
                count++;
            }
        }
        return count;
    }
}
```



```java
import junit.framework.TestCase;

public class ScoreEntryTest extends TestCase {

    FixedDice fd = new FixedDice("");

    public ScoreEntryTest(String name) {
        super(name);
    }

    protected void setUp() throws Exception {
        super.setUp();
    }

    protected void tearDown() throws Exception {
        super.tearDown();
    }

    /*
     * Test method for 'ScoreEntry.countN(DiceGroup, int)'
     */
    public void testCountN() {
        fd.set("12345");
        assertEquals(1,ScoreEntry.countN(fd,1));
        assertEquals(1,ScoreEntry.countN(fd,2));
        assertEquals(1,ScoreEntry.countN(fd,2));
        assertEquals(0,ScoreEntry.countN(fd,6));
        fd.set("11133");
        assertEquals(3,ScoreEntry.countN(fd,1));
        assertEquals(2,ScoreEntry.countN(fd,3));
        assertEquals(0,ScoreEntry.countN(fd,2));
        fd.set("33333");
        assertEquals(5,ScoreEntry.countN(fd,3));
        assertEquals(0,ScoreEntry.countN(fd,2));
        assertEquals(0,ScoreEntry.countN(fd,1));
    }

}
```

```java
public class SmallStraight extends ScoreEntry{
    SmallStraight()
    {

    }


    public String getDescription()
    {
        return "Small Straight (30)";
    }

    public int calcScore(DiceGroup dg)
    {
        return 0;
    }
}
```



```java
public class TestScoreEntry
{
    /**
     * Returns string representing all dice in dg
     * @param dg is the DiceGroup whose dice are returned
     * @return readable form of all dice in dg
     */

    public static String toString(DiceGroup dg)
    {
        String result = "";
        for(int k=0; k < dg.size(); k++) {
            result += dg.getValue(k);
        }
        return result;
    }

    /** pre: goodScore is the score that entry.calcScore(dg)
     *        is targetted to have. If entry doesn't have this
     *        target goodScore, then an error message is printed.
     */
    public static void testEntry(ScoreEntry entry,
                                 DiceGroup dg, int goodScore)
    {
        int score = entry.calcScore(dg);
        if (score != goodScore) {
            System.out.print("failed with " + entry.getDescription());
            System.out.print("dice = '" + toString(dg) + "' score = ");
            System.out.println(score+" TARGET = " + goodScore);
        }
    }

    public static void testOnes()
    {
        FixedDice d = new FixedDice("11222");
        ScoreEntry entry = new AboveLine(1);
        testEntry(entry,d,2);

        d.set("22222");
        testEntry(entry,d,0);

        d.set("11131");
        testEntry(entry,d,4);
    }

    public static void testTwos()
    {
        FixedDice d = new FixedDice("11222");
        ScoreEntry entry = new AboveLine(2);
        testEntry(entry,d,6);

        d.set("22222");
        testEntry(entry,d,10);

        d.set("11131");
        testEntry(entry,d,0);
    }

    public static void testChance()
    {
        FixedDice d = new FixedDice("11222");
        ScoreEntry entry = new Chance();
        testEntry(entry,d,8);

        d.set("66566");
        testEntry(entry,d,29);

        d.set("12345");
        testEntry(entry,d,15);
    }


    public static void testFourOfAKind()
```

```java
    {
        FixedDice d = new FixedDice("21212");
        ScoreEntry entry = new FourKind();
        testEntry(entry,d,0);

        d.set("34333");
        testEntry(entry,d,16);

        d.set("66166");
        testEntry(entry,d,25);

        d.set("55555");
        testEntry(entry,d,25);
    }


    public static void testLargeStraight()
    {
        FixedDice d = new FixedDice("12345");
    }

    public static void testFullHouse()
    {
        FixedDice d = new FixedDice("12121");
        ScoreEntry entry = new FullHouse();
        testEntry(entry,d,25);

        d.set("55333");
        testEntry(entry,d,25);

        d.set("11112");
        testEntry(entry,d,0);

        d.set("66666");
        testEntry(entry,d,0);
    }

    public static void testSmallStraight()
    {
        FixedDice d = new FixedDice("12345");
    }

    public static void main(String[] args)
    {
        testOnes();
        testTwos();
        testFourOfAKind();
        testSmallStraight();
        testLargeStraight();
        testChance();
        testFullHouse();
    }
}
```



```java
/**
 * This test program tests a CupOfDice to see
 * if the freeze/unfreezeAll functions work
 * together with rolling the dice in the group
 *
 * CupOfDice objects of size 1--6 (inclusive) are created.
 * To test a cup/group, each dice is rolled once, the roll
 * recorded, and the dice frozen
 *
 * Then the dice are unfrozen and rolled 100 times
 * to see if something different is rolled for each dice in
 * the group.
 *
 * If unfreezing works, the probability that something different
 * is *NOT* rolled is 1/(6^100) which is astronomically slim
 *
 * @author Owen Astrachan
 */

import java.util.Arrays;

public class TestUnfreeze
{
    private final static int NUMROLLS = 100;

    public static void test(int gsize)
    {
        CupOfDice dg = new CupOfDice(gsize);
        int j,k;
        int[]      rolls = new int[gsize];
        boolean[] passed = new boolean[gsize];
        Arrays.fill(passed,false);    // nothing passed

        // roll group and store each dice value, then freeze the dice

        dg.roll();
        for(k=0; k < gsize; k++) {
            rolls[k] = dg.getValue(k);
            dg.freeze(k);
        }

        /* now roll and test that dice aren't frozen
         * We do this by rolling dice and looking for a value
         * different from the first/originally rolled value
         * for each dice in the DiceGroup
         */

        for(k=0; k < NUMROLLS; k++) {
            dg.unfreezeAll();
            dg.roll();
            for(j=0; j < gsize; j++) {
                if (dg.getValue(j) != rolls[j]) {
                    passed[j] = true;
                }
            }
        }

        System.out.println("checking unfreeze on group size "+gsize);
        for(k=0; k < gsize; k++) {
            System.out.print("dice " + k + " ");
            if (passed[k]){
                System.out.println("passed");
            }
            else {
                System.out.println("failed");
            }
        }
    }

    public static void main(String[] args)
    {
        for(int k=1; k < 7; k++) {
```

```java
                TestUnfreeze.test(k);
            }
        }
    }
```



```java
import java.util.Observer;
import java.util.Observable;

public class TextDiceViewer implements Observer
{
    /**
     * Construct a text viewer.
     */

    public TextDiceViewer()
    {

    }

    /**
     * Displays all values of dg, typically not called
     * by client called, but via update and observer/observable.
     */

    public  void display(DiceGroup dg)
    {
        int k;
        for(k = 0; k < dg.size(); k++) {
            System.out.print("+---+\t");
        }
        System.out.println();

        for(k = 0; k < dg.size(); k++) {
            System.out.print("| " + dg.getValue(k) + " |\t");
        }
        System.out.println();

        for(k = 0; k < dg.size(); k++) {
            System.out.print("+---+\t");
        }
        System.out.println();

        for(k=0; k < dg.size(); k++) {
            System.out.print("  "+ k + "\t");
        }

        System.out.println();
        System.out.println();
        System.out.println();
    }

    /**
     * Called by observable, which must be a DiceGroup.
     * Prints values shown on dgroup.
     * @param dgroup is the dice group Observable.
     * @param dummy is ignored, required of Observer interface
     */

    public void update(Observable dgroup, Object dummy)
    {
        display((DiceGroup) dgroup);
    }

    public static void main(String[] args)
    {
        CupOfDice dice = new CupOfDice(5);
        TextDiceViewer viewer = new TextDiceViewer();
        dice.addObserver(viewer);

        dice.roll();
        dice.roll();
        dice.freeze(0);
        dice.freeze(1);
        dice.roll();
        dice.roll();
    }
}
```

```
import java.io.*;

public class TextScorer
{
    private ScoreCard myCard;
    private ScoreEntry[] myEntries;

    public TextScorer(ScoreCard s)
    {
        myCard = s;
        myEntries = myCard.getEntries();
    }

    /**
     * Prompt for an entry and score the DiceGroup dg.
     * precondition: scorecard isn't full
     */
    public void score(DiceGroup dg)
    {
        int num;
        String line = "";
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

        display();

        while (true){
            System.out.print("score by which number ");
            try{
                line = in.readLine();
            }
            catch(IOException ioe){
                ioe.printStackTrace();
                System.exit(0);
            }
            try{
                num = Integer.parseInt(line);
                num = num-1; // turn into vector index, start at 0
                if (num < 0 || num > 12){
                    System.out.println("Invalid scoring entry.");
                    System.out.println("Please try again.");
                }
                else if (myEntries[num].getScore() != -1){
                    System.out.println("That entry has already been scored");
                    System.out.println("Please try again.");
                }
                else{
                    break;
                }
            }
            catch(NumberFormatException nfe){
                System.out.println("Invalid scoring entry.");
                System.out.println("Please try again.");
            }
        }
        myEntries[num].scoreIt(dg);
    }

    /**
     * Displays each score entry and the current value for it.
     */
    public void display()
    {
        for(int k = 0; k < myEntries.length; k++) {
            int score = myEntries[k].getScore();
            System.out.print(k+1 + ".\t" +
                                 myEntries[k].getDescription()
                                 + "\t");
            if (score == -1) {
                System.out.println("*");
            }
            else {
                System.out.println(score);
```



```
            }
            if (k == 5) {
                System.out.println("---------");
                System.out.println();    // above the line done
            }
        }
        System.out.println();
    }


    /**
     * Displays each score entry, its value, and the total score of this
     *    ScoreCard object.
     */
    public void finalDisplay(){
        System.out.println("/**********************************/");
        System.out.println();
        display();
        System.out.println("/**********************************/");
        System.out.println("TOTAL SCORE = " + myCard.totalScore());
        System.out.println();
    }

}
```

```java
public class TextYahtzee
{
    public static void main(String[] args)
    {
        CupOfDice d5 = new CupOfDice(5);
        GroupRoller gr = new GroupRoller();
        ScoreCard sc = new ScoreCard();
        TextScorer scorer = new TextScorer(sc);
        TextDiceViewer viewer = new TextDiceViewer();
        d5.addObserver(viewer);

        while (! sc.allFilled())
        {
            scorer.display();
            gr.roll(d5,3);
            scorer.score(d5);
        }
    }
}
```



```java
public class ThreeKind extends ScoreEntry
{
    ThreeKind()
    {

    }

    public String getDescription()
    {
        return "3 of a kind (all)";
    }

    public int calcScore(DiceGroup dg)
    {
        return 0;
    }

}
```