```java
import ContinuousAssessment.Card;
import jdk.jfr.Description;

import static org.junit.jupiter.api.Assertions.assertEquals;

/**
 * Test class done using JUnit 4.13.1
 *
 * @author James Calnan & Nicholas Alexander
 * @version 1.0
 */
public class CardTest {

    @org.junit.Test
    @Description("Test to check the value of the card being stored in the object")
    public void testGetValue() {
        // Initialise a new card with a value 1
        Card c = new Card(1);

        // Test the value stored in the card object
        assertEquals(1, c.getValue());
    }

    @org.junit.Test
    @Description("Test to check the value of the ToString method")
    public void testToString() {
        // Initialise a new card with a value 1
        Card c = new Card(1);

        // Test the value of the ToString method
        assertEquals("1", String.valueOf(c));
    }
}
```

```java
import jdk.jfr.Description;

import java.io.*;
import java.util.ArrayList;
import java.util.Random;

import static org.junit.jupiter.api.Assertions.assertEquals;

/**
 * Test class done using JUnit 4.13.1
 *
 * @author James Calnan & Nicholas Alexander
 * @version 1.0
 */
public class PackTest {

    Pack testPack;
    ArrayList<Integer> temp = new ArrayList<>();


    /**
     * Method to initialise a new text file with 32 numbers in it
     * and then initialises a new Pack object with 4 players
     * @throws IOException
     */
    public void initFile() throws IOException {
        Random r = new Random();
        r.setSeed(54367853);

        String saveLocation = "testPack.txt";

        BufferedWriter bWriter = new BufferedWriter(new FileWriter(saveLocation));
        for (int i = 1; i <= 32; i++) {
            int val = r.nextInt(32);
            bWriter.write(val + "\n");
            temp.add(val);
        }

        bWriter.close();
        BufferedReader bReader = new BufferedReader(new FileReader(saveLocation));

        testPack = new Pack(4, bReader, saveLocation);
        bReader.close();
    }

    @org.junit.Test
    @Description("Test the value of the ToString method")
    public void testToString() throws IOException {
        // Initialise the pack file
        initFile();

        String cardsStr = "";
```

```
53          for (int c: temp) {
54              cardsStr += c+" ";
55          }
56
57          // Check the value
58          assertEquals(String.valueOf(testPack), "Pack card values: "
    + cardsStr);
59      }
60
61  }
```

```java
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.RunWith;
import org.junit.runner.notification.Failure;
import org.junit.runners.Suite;

import java.io.OutputStream;
import java.io.PrintStream;

@RunWith(Suite.class)

@Suite.SuiteClasses({
        CardTest.class,
        CardDeckTest.class,
        PackTest.class,
        PlayerTest.class
})

/**
 * Test class done using JUnit 4.13.1
 *
 * @author James Calnan & Nicholas Alexander
 * @version 1.0
 */
public class RunTests {
    public static void main(String[] args) {
        // Notify user that tests are being run
        System.out.println("Running tests");

        // Capture any outputs not relevant
        PrintStream originalStream = System.out;
        PrintStream dummyStream = new PrintStream(new OutputStream(){
public void write(int b) {} });
        System.setOut(dummyStream);

        // Run tests
        Result testResults = JUnitCore.runClasses(RunTests.class);

        // Reassign original print stream
        System.setOut(originalStream);

        System.out.println("\n\nTest Results: ");

        // Notify user of failed tests
        System.out.println(testResults.getFailures().size() + "
failures");
        for (Failure f : testResults.getFailures()) {
            System.out.println(f.toString());
        }

        System.out.println("\nTests successful: " + testResults.
wasSuccessful());
    }
}
```

```java
import ContinuousAssessment.Card;
import jdk.jfr.Description;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.LinkedList;
import java.util.Queue;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Test class done using JUnit 4.13.1
 *
 * @author James Calnan & Nicholas Alexander
 * @version 1.0
 */
public class PlayerTest {


    Queue<Card> tempCards = new LinkedList<>();

    /**
     * Method to initialise a new player
     * @return a new player object with 3 decks
     */
    Player createPlayer() {
        CardDeck playerDeck = new CardDeck(1);
        CardDeck lDeck = new CardDeck(2);
        CardDeck rDeck = new CardDeck(3);
        for (Card c: this.tempCards) {
            playerDeck.placeCardOnBottom(c);
            lDeck.placeCardOnBottom(c);
            rDeck.placeCardOnBottom(c);
        }
        return new Player(playerDeck, lDeck, rDeck);
    }

    @org.junit.Test
    @Description("Test to check the drawCardFromDeck method works")
    public void testDrawCardFromDeck() {
        // Clear tempCards ArrayList
        this.tempCards.clear();

        // Add new cards to the ArrayList
        for (int i = 4; i > 0; i--) {this.tempCards.add(new Card(i));}

        // Initialise a new player object
        Player p = createPlayer();

        // Record the value of the player on initialisation
        String valueBefore = String.valueOf(p);

```

```
55          // Draw a card from the deck
56          p.drawCardFromDeck();
57
58          // Define expected output
59          String expectedOutput = """
60
61                  Player number  : 1
62                  Player deck    : 1 2 3 4 4\s
63                  Deck no 2 ldeck: 3 2 1\s
64                  Deck no 3 rdeck: 4 3 2 1\s
65                  """;
66
67          // Define actual output
68          String actualOutput = String.valueOf(p);
69
70          // Assertion tests
71          assertEquals(expectedOutput, actualOutput);
72          assertNotEquals(valueBefore, actualOutput);
73      }
74
75      @org.junit.Test
76      @Description("Test to check the cards are being discarded
    correctly")
77      public void testDiscardCardToDeck() {
78          // Clear tempCards ArrayList
79          this.tempCards.clear();
80
81          // Add new cards to the ArrayList
82          for (int i = 4; i > 0; i--) {this.tempCards.add(new Card(i
    ));}
83
84          // Initialise a new player object
85          Player p = createPlayer();
86
87          // Record the value of the player on initialisation
88          String valueBefore = String.valueOf(p);
89
90          // Discard a card from the players deck
91          p.discardCardToDeck();
92
93          // Define expected output
94          String expectedOutput = """
95
96                  Player number  : 1
97                  Player deck    : 1 2 3\s
98                  Deck no 2 ldeck: 4 3 2 1\s
99                  Deck no 3 rdeck: 4 3 2 1 4\s
100                 """;
101
102         // Define actual output
103         String actualOutput = String.valueOf(p);
104
105         // Actual output
106         assertEquals(expectedOutput, actualOutput);
```

```
107         assertNotEquals(valueBefore, actualOutput);
108 
109     }
110 
111     @org.junit.Test
112     @Description("Test to simulate the playGo method")
113     public void testPlayGo() {
114         // Clear tempCards ArrayList
115         this.tempCards.clear();
116 
117         // Add new cards to the ArrayList
118         for (int i = 4; i > 0; i--) {this.tempCards.add(new Card(i
    ));}
119 
120         // Initialise a new player object
121         Player p = createPlayer();
122 
123         // Simulate a round
124         p.playGo();
125 
126         // Define the expected output
127         String expectedOutput = """
128 
129                 Player number  : 1
130                 Player deck    : 1 2 3 4\s
131                 Deck no 2 ldeck: 4 3 2 1\s
132                 Deck no 3 rdeck: 4 3 2 1\s
133                 """;
134 
135         // Define the actual output
136         String actualOutput = String.valueOf(p);
137 
138         // Assertion test
139         assertEquals(expectedOutput, actualOutput);
140     }
141 
142     @org.junit.Test
143     @Description("Test to check the ToString method is outputting the
     correct message")
144     public void testToString() {
145         // Clear tempCards ArrayList
146         this.tempCards.clear();
147 
148         // Add new cards to the ArrayList
149         for (int i = 4; i > 0; i--) {this.tempCards.add(new Card(1
    ));}
150 
151         // Initialise a new player object
152         Player p = createPlayer();
153 
154         // Define the expected output
155         String expectedOutput = "\nPlayer number  : 1\nPlayer deck
        : 1 1 1 1 \nDeck no 2 ldeck: 1 1 1 1 \nDeck no 3 rdeck: 1 1 1 1 \
    n";
```

```
156
157         // Define the actual output
158         String actualOutput = String.valueOf(p);
159
160         // Assertion test
161         assertEquals(actualOutput, expectedOutput);
162     }
163
164     @org.junit.Test
165     @Description("Test to check the getPlayerWon method is working
    correctly")
166     public void testAllSameCards() {
167         // Clear tempCards ArrayList
168         this.tempCards.clear();
169
170         // Add new cards to the ArrayList
171         for (int i = 4; i > 0; i--) {this.tempCards.add(new Card(1
    ));}
172
173         // Initialise a new player object
174         Player p = createPlayer();
175
176         // Check the player has won
177         assertTrue(p.getPlayerWon());
178     }
179
180
181     @org.junit.Test
182     @Description("Test to check the correct messages are being
    outputted to the text file")
183     public void testLogOutput() throws IOException {
184         // Clear tempCards ArrayList
185         this.tempCards.clear();
186
187         // Add new cards to the ArrayList
188         for (int i = 4; i > 0; i--) {this.tempCards.add(new Card(i
    ));}
189
190         // Initialise a new player object
191         Player p = createPlayer();
192
193         // Draw a card from the deck
194         p.drawCardFromDeck();
195
196         // Define the file reader
197         BufferedReader reader = new BufferedReader(new FileReader("
    Logs" + File.separator + "player1_output.txt"));
198
199         // Record the first and second line of the text file
200         reader.readLine();
201         String value2 = reader.readLine();
202
203         assertEquals("player 1 draws a 4 from deck 2", value2);
204     }
```

```
205
206
207     @org.junit.Test
208     @Description("Test to check that the correct lose output message
    is being saved to the text file")
209     public void testLoseOutput() throws IOException {
210         // Clear tempCards ArrayList
211         this.tempCards.clear();
212
213         // Add new cards to the ArrayList
214         for (int i = 4; i > 0; i--) {this.tempCards.add(new Card(i
    ));}
215
216         // Initialise a new player object
217         Player p = createPlayer();
218
219         // Output the lost message to the text file
220         p.loseOutput();
221
222         // Define the file reader
223         BufferedReader reader = new BufferedReader(new FileReader("
    Logs" + File.separator + "player1_output.txt"));
224         // Record the first and second line of the text file
225         reader.readLine();
226         String value2 = reader.readLine();
227
228         // Assertion test
229         assertEquals("player 1 has informed player 1 that player 1
    has won", value2);
230     }
231
232     @org.junit.Test
233     @Description("Test to check the correct win output message is
    being saved in the text file")
234     public void testWinOutput() throws IOException {
235         // Clear tempCards ArrayList
236         this.tempCards.clear();
237
238         // Add new cards to the ArrayList
239         for (int i = 4; i > 0; i--) {this.tempCards.add(new Card(i
    ));}
240
241         // Initialise a new player object
242         Player p = createPlayer();
243
244         // Output the win message to the text file
245         p.winOutput();
246
247         // Define the file reader
248         BufferedReader reader = new BufferedReader(new FileReader("
    Logs" + File.separator + "player1_output.txt"));
249         // Record the first and second line of the text file
250         reader.readLine();
251         String value2 = reader.readLine();
```

```
252
253         // Assertion test
254         assertEquals("player 1 wins", value2);
255     }
256
257     @org.junit.Test
258     @Description("Test to check the current hand output is correct")
259     public void testCurrentHandOutput() {
260         // Clear tempCards ArrayList
261         this.tempCards.clear();
262
263         // Add new cards to the ArrayList
264         for (int i = 4; i > 0; i--) {this.tempCards.add(new Card(i
    ));}
265
266         // Initialise a new player object
267         Player p = createPlayer();
268
269         // Assertion test
270         assertEquals("1 2 3 4 ", p.getOrderedCards());
271     }
272
273     @org.junit.Test
274     @Description("Test to check that the playerWon method works when
    the deck satisfies the requirements")
275     public void testSetPlayerWon() {
276         // Initialise three temporary decks
277         CardDeck temp1 = new CardDeck(1);
278         CardDeck temp2 = new CardDeck(2);
279         CardDeck temp3 = new CardDeck(3);
280
281         // Add an extra card to the players deck so that they don't
    win on initialisation
282         temp1.placeCardOnBottom(new Card(2));
283
284         // Populate the decks with cards
285         for (int i = 0; i < 4; i++) {
286             temp1.placeCardOnBottom(new Card(1));
287             temp2.placeCardOnBottom(new Card(1));
288             temp3.placeCardOnBottom(new Card(1));
289         }
290
291         // Initialise a new player object
292         Player p = new Player(temp1, temp2, temp3);
293
294         // Draw and discard a card
295         p.discardCardToDeck();
296         p.drawCardFromDeck();
297
298         // Check if the player has won
299         boolean checkWin = p.getPlayerWon();
300
301         // Set if the player has won
302         p.setPlayerWon();
```

```
303
304         // Assertion test
305         assertNotEquals(checkWin, p.getPlayerWon());
306
307     }
308 }
```

```java
import jdk.jfr.Description;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertNotEquals;

/**
 * Test class done using JUnit 4.13.1
 *
 * @author James Calnan & Nicholas Alexander
 * @version 1.0
 */
public class CardDeckTest {


    /**
     * @return new deck with cards in it
     */
    CardDeck initDeck() {
        // Initialise new deck object with a deckId of 1
        CardDeck d = new CardDeck(1);

        // Populate the deck with cards using the placeCardOnBottom method
        for (int i = 1; i < 5; i++) {d.placeCardOnBottom(new ContinuousAssessment.Card(i));}

        // Return the deck
        return d;
    }

    @org.junit.Test
    @Description("Test to check cards are being taken from the top of the correct deck")
    public void testTakeCardFromTop() {
        // Initialise a new deck for testing
        CardDeck testDeck = initDeck();

        // Record value of the deck on initialisation
        String valueBefore = String.valueOf(testDeck);

        // Take a card from the top of the deck
        testDeck.takeCardFromTop();

        // Define the expected output and the actual output
        String expectedOutput = "2 3 4 ";
        String actualOutput = String.valueOf(testDeck);

        // Assertion tests
        assertEquals(expectedOutput, actualOutput);
        assertNotEquals(valueBefore, actualOutput);
    }

    @org.junit.Test
    @Description("Test to check cards are being placed on the bottom
```

```
of the correct deck")
    public void testPlaceCardOnBottom() {
        // Initialise a new deck for testing
        CardDeck testDeck = initDeck();

        // Record value of the deck on initialisation
        String valueBefore = String.valueOf(testDeck);

        // Place a new card of the bottom of the deck
        testDeck.placeCardOnBottom(new ContinuousAssessment.Card(10
));

        // Define the expected output and the actual output
        String expectedOutput = "1 2 3 4 10 ";
        String actualOutput = String.valueOf(testDeck);

        // Assertion tests
        assertEquals(expectedOutput, actualOutput);
        assertNotEquals(valueBefore, actualOutput);
    }

    @org.junit.Test
    @Description("Test to check the ToString method output the
correct data")
    public void testToString() {
        // Initialise a new deck for testing
        CardDeck testDeck = initDeck();

        // Assertion test to check the value of the deck
        assertEquals(String.valueOf(testDeck), "1 2 3 4 ");
    }

}
```