```
 1 package catan;
 2
 3 import java.awt.Color;
12
13 /**
14  * This class holds the methods and variables available to Players
   in the Catan board game.
15  *
16  * @author Eddie Gurnee
17  * @author Nicole Downer
18  * @version 0.0.14 10/28/2013
19  *
20  */
21 public class Player {
22     /**
23      * This class holds the data for the labels showing how many
   pieces that remain.
24      *
25      * @author Eddie Gurnee
26      * @version 0.0.05 10/29/2013
27      *
28      */
29     private class PieceLabel extends JPanel {
30         private String name;
31         private int type;
32
33         private final int WIDTH = 160;
34         private final int HEIGHT = 25;
35         private PieceLabel(int type) {
36             super();
37             setSize(WIDTH, HEIGHT);
38             this.setMinimumSize(getSize());
39
40             setOpaque(true);
41
42             this.type = type;
43             if (type == 0) {
44                 this.name = "Roads: ";
45             } else if (type == 1) {
46                 this.name = "Settlements: ";
47             } else if (type == 2) {
```

```
48                  this.name = "Cities: ";
49              }
50          }
51          public void paintComponent(Graphics g) {
52              super.paintComponent(g);
53
54              String str = this.name;
55
56              switch (this.type) {
57              case 0:
58                  str += Player.this.getRemainingRoads();
59                  break;
60              case 1:
61                  str += Player.this.getRemainingSettlements();
62                  break;
63              case 2:
64                  str += Player.this.getRemainingCities();
65                  break;
66              }
67
68              g.setColor(Player.this.getColor());
69              g.setFont(new Font(Font.SANS_SERIF, Font.BOLD, 18));
70
71              g.drawString(str, 10, HEIGHT);
72          }
73      }
74      //initial player variables
75      private String name;
76      private String colorName;
77      private Color color;
78
79      //do they have the longest road or largest army
80      private boolean longestRoad;
81      private boolean largestArmy;
82
83      //starting variables
84      private boolean theRoadPlaced;
85      private boolean secondRoadPlaced;
86
87      //the length of the two roads
88      private int road1;
```

```
89      private int road2;
90
91      //the size of the army
92      private int army = 0;
93
94      //used to keep track of the free roads from road building
95      private int roadBuilding;
96
97      //Wood, Sheep, Wheat, Ore, Brick
98      private int[] resources = new int[5];
99      //the resources gain while it wasn't their turn
100     private int[] tempResources = new int[5];
101     //Road, Settlement, City
102     private int[] pieces = new int[3];
103
104     //max number of pieces per player
105     private final int MAX_CITIES = 4;
106     private final int MAX_SETTLEMENTS = 5;
107     private final int MAX_ROADS = 15;
108
109     //what cards do they currently own
110     private ArrayList<DevCard> devCards = new ArrayList<>();
111     private boolean devCardPlayed = false;
112
113     //the corresponding piece labels for the player
114     private PieceLabel[] pieceLabels = new PieceLabel[3];
115
116     public Player() {
117         this("no name","no color");
118     }
119     public Player(String name, String colorName) {
120         this.name = name;
121         this.colorName = colorName;
122         this.theRoadPlaced = false;
123         this.roadBuilding = 0;
124
125         for (int i = 0; i < pieceLabels.length; i++) {
126             pieceLabels[i] = new PieceLabel(i);
127         }
128
129         switch (colorName) {
```

```
130         case "Red":
131             this.color = Color.RED;
132             break;
133         case "Blue":
134             this.color = Color.BLUE;
135             break;
136         case "Orange":
137             this.color = Color.ORANGE;
138             break;
139         case "White":
140             this.color = Color.WHITE;
141             break;
142         default:
143             this.color = Color.BLACK;
144             break;
145         }
146     }
147     public void displayNewResources() {
148         String[] types = {"Wood", "Sheep", "Wheat", "Ore", "Brick"};
149 
150         boolean display = false;
151         for (int x : tempResources) {
152             if (x > 0) {
153                 display = true;
154             }
155         }
156         String str = this + " you produced";
157         if (display) {
158             str += ":";
159             for (int i = 0; i < tempResources.length; i++) {
160                 if (tempResources[i] > 0) {
161                     str += "\n" + tempResources[i] + " " + types[i];
162                 }
163             }
164             for (int i = 0; i < tempResources.length; i++) {
165                 this.resources[i] += this.tempResources[i];
166             }
167             this.resetTempResource();
168         } else {
169             str += " nothing!\nBad luck!";
170         }
```

```
171         JOptionPane.showMessageDialog
172         (null,
173                 str,
174                 "Your new resources:",
175                 JOptionPane.PLAIN_MESSAGE);
176     }
177     public void displayCurrentResources() {
178         JOptionPane.showMessageDialog
179         (null,
180                 "Wood: " + resources[0] +
181                 "\nSheep: " + resources[1] +
182                 "\nWheat: " + resources[2] +
183                 "\nOre: " + resources[3] +
184                 "\nBrick: " + resources[4] ,
185                 "Your current resources:",
186                 JOptionPane.PLAIN_MESSAGE);
187     }
188     public void displayTempResources() {
189         String[] types = {"Wood", "Sheep", "Wheat", "Ore", "Brick"};
190 
191         boolean display = false;
192         for (int x : tempResources) {
193             if (x > 0) {
194                 display = true;
195             }
196         }
197         if (display) {
198             String str = this + " you made:";
199             for (int i = 0; i < tempResources.length; i++) {
200                 if (tempResources[i] > 0) {
201                     str += "\n" + tempResources[i] + " " + types[i];
202                 }
203             }
204             str += "\nOn the other players turns.";
205             JOptionPane.showMessageDialog
206             (null,
207                     str,
208                     "Your new resources:",
209                     JOptionPane.PLAIN_MESSAGE);
210             for (int i = 0; i < tempResources.length; i++) {
211                 this.resources[i] += this.tempResources[i];
```

```
212             }
213             this.resetTempResource();
214         }
215     }
216     private void resetTempResource() {
217         for (int i = 0; i < tempResources.length; i++) {
218             this.tempResources[i] = 0;
219         }
220     }
221     public void addTempResource(int resource, int amount) {
222         BoardGame.bank[resource] -= amount;
223         this.tempResources[resource] += amount;
224     }
225     public void addStartResources() {
226         addResource(0, 2);
227         addResource(1, 1);
228         addResource(2, 1);
229         addResource(3, 0);
230         addResource(4, 2);
231     }
232     public void stealResource(int resource, int amount) {
233         this.resources[resource] += amount;
234     }
235     public void addResource(int resource, int amount) {
236         BoardGame.bank[resource] -= amount;
237         this.resources[resource] += amount;
238     }
239     public void robberDiscard() {
240         int numCards = 0;
241         for (int r : resources) {
242             numCards += r;
243         }
244         if (numCards > 7) {
245             String[] theResources = {"Wood", "Sheep", "Wheat",
    "Ore", "Brick"};
246             for (int j = 0; j < (numCards - 1) / 2; j++) {
247                 ArrayList<String> reTemp = new ArrayList<>();
248                 for (int i = 0; i < resources.length; i++) {
249                     if (resources[i] > 0) {
250                         reTemp.add(theResources[i]);
251                     }
```

```
252                 }
253                 int re = JOptionPane.showOptionDialog
254                         (null,
255                                 "Select a resource to give up:",
256                                 "Robber Rolled:",
257                                 JOptionPane.YES_NO_CANCEL_OPTION,
258                                 JOptionPane.PLAIN_MESSAGE,
259                                 null,
260                                 reTemp.toArray(),
261                                 reTemp.get(0)
262                                 );
263 
264 BoardGame.bank[Arrays.asList(theResources).indexOf(reTemp.get(re))] += 1;
265 this.resources[Arrays.asList(theResources).indexOf(reTemp.get(re))] -= 1;
266             }
267         }
268     }
269     public int monopoly(int resource) {
270         int amount = this.resources[resource];
271         this.resources[resource] -= amount;
272 
273         return amount;
274     }
275     public void setRoadBuilding() {
276         this.roadBuilding += 2;
277     }
278     public boolean isRoadBuilding() {
279         return this.roadBuilding != 0;
280     }
281     public void builtFreeRoad() {
282         this.roadBuilding--;
283     }
284     public void buyDevCard() throws NotEnoughResourcesException, NotEnoughCardsException {
285         if (!this.isEnoughMoney(DevCard.getCost())) {
286             throw new NotEnoughResourcesException("Development Card");
```

```
287         }
288         if (BoardGame.devCardsBank.size() == 0) {
289             throw new NotEnoughCardsException("Development");
290         }
291         devCards.add(BoardGame.devCardsBank.get(0));
292         JOptionPane.showMessageDialog
293         (null,
294                 "You got a " +
    BoardGame.devCardsBank.get(0).toString() + " card.",
295                 "Your new development card:",
296                 JOptionPane.PLAIN_MESSAGE);
297         BoardGame.devCardsBank.remove(0);
298         for (int i = 0; i < this.resources.length; i++) {
299             this.resources[i] -= DevCard.getCost()[i];
300             BoardGame.bank[i] += DevCard.getCost()[i];
301         }
302     }
303     public void playDevCard() {
304         if (devCards.size() != 0) {
305             if (devCardPlayed) {
306                 JOptionPane.showMessageDialog
307                 (null,
308                         "You can only play one development card per
    turn!",
309                         "",
310                         JOptionPane.PLAIN_MESSAGE);
311             } else {
312                 DevCard[] devCardArr = new DevCard[devCards.size()];
313                 devCardArr = devCards.toArray(devCardArr);
314 
315                 DevCard selected =
    (DevCard)JOptionPane.showInputDialog
316                         (null,
317                                 "Which of your cards would you like
    to play?",
318                                 "Play Development Cards",
319                                 JOptionPane.PLAIN_MESSAGE,
320                                 null,
321                                 devCardArr,
322                                 devCardArr[0]);
323                 try {
```

```
324                     selected.play();
325                     devCards.remove(devCards.indexOf(selected));
326                     BoardGame.devCardsBank.add(selected);
327                 } catch (NullPointerException ex) {
328                     //Don't you worry child
329                 }
330             }
331         } else {
332             JOptionPane.showMessageDialog
333             (null,
334                     "You currently have no development cards!",
335                     "",
336                     JOptionPane.PLAIN_MESSAGE);
337         }
338     }
339     public boolean isEnoughMoney(int[] payCost) {
340         boolean enough = true;
341         for (int i = 0; i < resources.length; i++) {
342             if (resources[i] - payCost[i] < 0) {
343                 enough = false;
344             }
345         }
346         return enough;
347     }
348     public int getRemainingRoads() {
349         return MAX_ROADS - pieces[0];
350     }
351     public int getRemainingSettlements() {
352         return MAX_SETTLEMENTS - pieces[1];
353     }
354     public int getRemainingCities() {
355         return MAX_CITIES - pieces[2];
356     }
357     public boolean isRemainingRoads() {
358         return (pieces[0] < MAX_ROADS);
359     }
360     public boolean isRemainingSettlements() {
361         return (pieces[1] < MAX_SETTLEMENTS);
362     }
363     public boolean isRemainingCities() {
364         return (pieces[2] < MAX_CITIES);
```

```
365     }
366     public void buyPiece(Piece thePiece) {
367         for (int i = 0; i < this.resources.length; i++) {
368             this.resources[i] -= thePiece.getCost()[i];
369             BoardGame.bank[i] += thePiece.getCost()[i];
370         }
371         if (thePiece.getClass() == Road.class) {
372             this.placedRoad();
373         } else if (thePiece.getClass() == Properties.class) {
374             if (((Properties)thePiece).getCity()) {
375                 this.placedCity();
376             } else {
377                 this.placedSettlement();
378             }
379         }
380         for (PieceLabel p : pieceLabels) {
381             p.repaint();
382         }
383     }
384     public void placedSettlement() {
385         this.pieces[1]++;
386     }
387     public void placedCity() {
388         this.pieces[2]++;
389         this.pieces[1]--;
390     }
391     public void placedRoad() {
392         this.pieces[0]++;
393     }
394     public void setLargestArmy(boolean largestArmy) {
395         this.largestArmy = largestArmy;
396     }
397     public int getVictoryPoints() {
398         int vp = (pieces[1] * 1) + (pieces[2] * 2);
399         if (longestRoad) {
400             vp += 2;
401         }
402         if (largestArmy) {
403             vp += 2;
404         }
405         for (DevCard d : devCards) {
```

```
406             vp += d.getVictoryPoints();
407         }
408         return vp;
409     }
410     public Color getColor() {
411         return color;
412     }
413     public String getName() {
414         return name;
415     }
416     public boolean isTheRoadPlaced() {
417         return theRoadPlaced;
418     }
419     public void setTheRoadPlaced(boolean theRoadPlaced) {
420         this.theRoadPlaced = theRoadPlaced;
421     }
422     public boolean isSecondRoadPlaced() {
423         return secondRoadPlaced;
424     }
425     public void setSecondRoadPlaced(boolean secondRoadPlaced) {
426         this.secondRoadPlaced = secondRoadPlaced;
427     }
428     @Override
429     public String toString() {
430         return (this.colorName + " player: " + this.name);
431     }
432     @Override
433     public boolean equals(Object otherObject) {
434         if (otherObject == null) {
435             return false;
436         } else if (this.getClass() != otherObject.getClass()) {
437             return false;
438         } else {
439             Player otherPlayer = (Player)otherObject;
440             return (this.color.equals(otherPlayer.color)
441                     && this.name.equals(otherPlayer.name)
442                     && this.colorName.equals(otherPlayer.colorName)
443                     );
444         }
445     }
446     public int getArmy() {
```

```
447         return army;
448     }
449     public void addArmy() {
450         this.army++;
451     }
452     public Component getPieceLabel(int i) {
453         return pieceLabels[i];
454     }
455 
456     //Method to use devCard(type) can only use one at a time
457     //remove from array
458     //activate item chosen (increase VP, add knight, or progress
    card(multiple types))
459 
460     //Method to determine largest army
461     //if (knights cards in play >= 3) true
462 
463     //Method to determine longest road
464     // if (roadCount > all other players) true
465 
466 }
```