```java
package catan;

import java.awt.Color;

/**
 * This class holds the methods and variables available to Players in the Catan board game.
 *
 * @author Eddie Gurnee
 * @author Nicole Downer
 * @version 0.0.14 10/28/2013
 *
 */
public class Player {
    /**
     * This class holds the data for the labels showing how many pieces that remain.
     *
     * @author Eddie Gurnee
     * @version 0.0.05 10/29/2013
     *
     */
    private class PieceLabel extends JPanel {
        private String name;
        private int type;

        private final int WIDTH = 160;
        private final int HEIGHT = 25;
        private PieceLabel(int type) {
            super();
            setSize(WIDTH, HEIGHT);
            this.setMinimumSize(getSize());

            setOpaque(true);

            this.type = type;
            if (type == 0) {
                this.name = "Roads: ";
            } else if (type == 1) {
                this.name = "Settlements: ";
            } else if (type == 2) {
                this.name = "Cities: ";
            }
        }
        public void paintComponent(Graphics g) {
            super.paintComponent(g);

            String str = this.name;

            switch (this.type) {
            case 0:
                str += Player.this.getRemainingRoads();
                break;
            case 1:
                str += Player.this.getRemainingSettlements();
                break;
            case 2:
                str += Player.this.getRemainingCities();
                break;
            }

            g.setColor(Player.this.getColor());
            g.setFont(new Font(Font.SANS_SERIF, Font.BOLD, 18));

            g.drawString(str, 10, HEIGHT);
        }
    }
    //initial player variables
    private String name;
    private String colorName;
    private Color color;

    //do they have the longest road or largest army
    private boolean longestRoad;
    private boolean largestArmy;

    //starting variables
    private boolean theRoadPlaced;
    private boolean secondRoadPlaced;

    //the length of the two roads
    private int road1;
    private int road2;

    //the size of the army
    private int army = 0;

    //used to keep track of the free roads from road building
    private int roadBuilding;

    //Wood, Sheep, Wheat, Ore, Brick
    private int[] resources = new int[5];
    //the resources gain while it wasn't their turn
    private int[] tempResources = new int[5];
    //Road, Settlement, City
    private int[] pieces = new int[3];

    //max number of pieces per player
    private final int MAX_CITIES = 4;
    private final int MAX_SETTLEMENTS = 5;
    private final int MAX_ROADS = 15;

    //what cards do they currently own
    private ArrayList<DevCard> devCards = new ArrayList<>();
    private boolean devCardPlayed = false;

    //the corresponding piece labels for the player
    private PieceLabel[] pieceLabels = new PieceLabel[3];

    public Player() {
        this("no name","no color");
    }
    public Player(String name, String colorName) {
        this.name = name;
        this.colorName = colorName;
        this.theRoadPlaced = false;
        this.roadBuilding = 0;

        for (int i = 0; i < pieceLabels.length; i++) {
            pieceLabels[i] = new PieceLabel(i);
        }

        switch (colorName) {
```

```java
        case "Red":
            this.color = Color.RED;
            break;
        case "Blue":
            this.color = Color.BLUE;
            break;
        case "Orange":
            this.color = Color.ORANGE;
            break;
        case "White":
            this.color = Color.WHITE;
            break;
        default:
            this.color = Color.BLACK;
            break;
        }
    }
    public void displayNewResources() {
        String[] types = {"Wood", "Sheep", "Wheat", "Ore", "Brick"};

        boolean display = false;
        for (int x : tempResources) {
            if (x > 0) {
                display = true;
            }
        }
        String str = this + " you produced";
        if (display) {
            str += ":";
            for (int i = 0; i < tempResources.length; i++) {
                if (tempResources[i] > 0) {
                    str += "\n" + tempResources[i] + " " + types[i];
                }
            }
            for (int i = 0; i < tempResources.length; i++) {
                this.resources[i] += this.tempResources[i];
            }
            this.resetTempResource();
        } else {
            str += " nothing!\nBad luck!";
        }
        JOptionPane.showMessageDialog
        (null,
                str,
                "Your new resources:",
                JOptionPane.PLAIN_MESSAGE);
    }
    public void displayCurrentResources() {
        JOptionPane.showMessageDialog
        (null,
                "Wood: " + resources[0] +
                "\nSheep: " + resources[1] +
                "\nWheat: " + resources[2] +
                "\nOre: " + resources[3] +
                "\nBrick: " + resources[4] ,
                "Your current resources:",
                JOptionPane.PLAIN_MESSAGE);
    }
    public void displayTempResources() {
        String[] types = {"Wood", "Sheep", "Wheat", "Ore", "Brick"};

        boolean display = false;
        for (int x : tempResources) {
            if (x > 0) {
                display = true;
            }
        }
        if (display) {
            String str = this + " you made:";
            for (int i = 0; i < tempResources.length; i++) {
                if (tempResources[i] > 0) {
                    str += "\n" + tempResources[i] + " " + types[i];
                }
            }
            str += "\nOn the other players turns.";
            JOptionPane.showMessageDialog
            (null,
                    str,
                    "Your new resources:",
                    JOptionPane.PLAIN_MESSAGE);
            for (int i = 0; i < tempResources.length; i++) {
                this.resources[i] += this.tempResources[i];
```

```java
            }
            this.resetTempResource();
        }
    }
    private void resetTempResource() {
        for (int i = 0; i < tempResources.length; i++) {
            this.tempResources[i] = 0;
        }
    }
    public void addTempResource(int resource, int amount) {
        BoardGame.bank[resource] -= amount;
        this.tempResources[resource] += amount;
    }
    public void addStartResources() {
        addResource(0, 2);
        addResource(1, 1);
        addResource(2, 1);
        addResource(3, 0);
        addResource(4, 2);
    }
    public void stealResource(int resource, int amount) {
        this.resources[resource] += amount;
    }
    public void addResource(int resource, int amount) {
        BoardGame.bank[resource] -= amount;
        this.resources[resource] += amount;
    }
    public void robberDiscard() {
        int numCards = 0;
        for (int r : resources) {
            numCards += r;
        }
        if (numCards > 7) {
            String[] theResources = {"Wood", "Sheep", "Wheat",
 "Ore", "Brick"};
            for (int j = 0; j < (numCards - 1) / 2; j++) {
                ArrayList<String> reTemp = new ArrayList<>();
                for (int i = 0; i < resources.length; i++) {
                    if (resources[i] > 0) {
                        reTemp.add(theResources[i]);
                    }
                }
                int re = JOptionPane.showOptionDialog
                        (null,
                                "Select a resource to give up:",
                                "Robber Rolled:",
                                JOptionPane.YES_NO_CANCEL_OPTION,
                                JOptionPane.PLAIN_MESSAGE,
                                null,
                                reTemp.toArray(),
                                reTemp.get(0)
                                );


BoardGame.bank[Arrays.asList(theResources).indexOf(reTemp.get(re))]
+= 1;
this.resources[Arrays.asList(theResources).indexOf(reTemp.get(re))]
-= 1;
            }
        }
    }
    public int monopoly(int resource) {
        int amount = this.resources[resource];
        this.resources[resource] -= amount;

        return amount;
    }
    public void setRoadBuilding() {
        this.roadBuilding += 2;
    }
    public boolean isRoadBuilding() {
        return this.roadBuilding != 0;
    }
    public void builtFreeRoad() {
        this.roadBuilding--;
    }
    public void buyDevCard() throws NotEnoughResourcesException,
NotEnoughCardsException {
        if (!this.isEnoughMoney(DevCard.getCost())) {
            throw new NotEnoughResourcesException("Development
Card");
```

```java
        }
        if (BoardGame.devCardsBank.size() == 0) {
            throw new NotEnoughCardsException("Development");
        }
        devCards.add(BoardGame.devCardsBank.get(0));
        JOptionPane.showMessageDialog
        (null,
                "You got a " + BoardGame.devCardsBank.get(0).toString() + " card.",
                "Your new development card:",
                JOptionPane.PLAIN_MESSAGE);
        BoardGame.devCardsBank.remove(0);
        for (int i = 0; i < this.resources.length; i++) {
            this.resources[i] -= DevCard.getCost()[i];
            BoardGame.bank[i] += DevCard.getCost()[i];
        }
    }
    public void playDevCard() {
        if (devCards.size() != 0) {
            if (devCardPlayed) {
                JOptionPane.showMessageDialog
                (null,
                        "You can only play one development card per turn!",
                        "",
                        JOptionPane.PLAIN_MESSAGE);
            } else {
                DevCard[] devCardArr = new DevCard[devCards.size()];
                devCardArr = devCards.toArray(devCardArr);

                DevCard selected = (DevCard)JOptionPane.showInputDialog
                        (null,
                                "Which of your cards would you like to play?",
                                "Play Development Cards",
                                JOptionPane.PLAIN_MESSAGE,
                                null,
                                devCardArr,
                                devCardArr[0]);
                try {
                    selected.play();
                    devCards.remove(devCards.indexOf(selected));
                    BoardGame.devCardsBank.add(selected);
                } catch (NullPointerException ex) {
                    //Don't you worry child
                }
            }
        } else {
            JOptionPane.showMessageDialog
            (null,
                    "You currently have no development cards!",
                    "",
                    JOptionPane.PLAIN_MESSAGE);
        }
    }
    public boolean isEnoughMoney(int[] payCost) {
        boolean enough = true;
        for (int i = 0; i < resources.length; i++) {
            if (resources[i] - payCost[i] < 0) {
                enough = false;
            }
        }
        return enough;
    }
    public int getRemainingRoads() {
        return MAX_ROADS - pieces[0];
    }
    public int getRemainingSettlements() {
        return MAX_SETTLEMENTS - pieces[1];
    }
    public int getRemainingCities() {
        return MAX_CITIES - pieces[2];
    }
    public boolean isRemainingRoads() {
        return (pieces[0] < MAX_ROADS);
    }
    public boolean isRemainingSettlements() {
        return (pieces[1] < MAX_SETTLEMENTS);
    }
    public boolean isRemainingCities() {
        return (pieces[2] < MAX_CITIES);
    }
    public void buyPiece(Piece thePiece) {
        for (int i = 0; i < this.resources.length; i++) {
            this.resources[i] -= thePiece.getCost()[i];
            BoardGame.bank[i] += thePiece.getCost()[i];
        }
        if (thePiece.getClass() == Road.class) {
            this.placedRoad();
        } else if (thePiece.getClass() == Properties.class) {
            if (((Properties)thePiece).getCity()) {
                this.placedCity();
            } else {
                this.placedSettlement();
            }
        }
        for (PieceLabel p : pieceLabels) {
            p.repaint();
        }
    }
    public void placedSettlement() {
        this.pieces[1]++;
    }
    public void placedCity() {
        this.pieces[2]++;
        this.pieces[1]--;
    }
    public void placedRoad() {
        this.pieces[0]++;
    }
    public void setLargestArmy(boolean largestArmy) {
        this.largestArmy = largestArmy;
    }
    public int getVictoryPoints() {
        int vp = (pieces[1] * 1) + (pieces[2] * 2);
        if (longestRoad) {
            vp += 2;
        }
        if (largestArmy) {
            vp += 2;
        }
        for (DevCard d : devCards) {
            vp += d.getVictoryPoints();
        }
        return vp;
    }
    public Color getColor() {
        return color;
    }
    public String getName() {
        return name;
    }
    public boolean isTheRoadPlaced() {
        return theRoadPlaced;
    }
    public void setTheRoadPlaced(boolean theRoadPlaced) {
        this.theRoadPlaced = theRoadPlaced;
    }
    public boolean isSecondRoadPlaced() {
        return secondRoadPlaced;
    }
    public void setSecondRoadPlaced(boolean secondRoadPlaced) {
        this.secondRoadPlaced = secondRoadPlaced;
    }
    @Override
    public String toString() {
        return (this.colorName + " player: " + this.name);
    }
    @Override
    public boolean equals(Object otherObject) {
        if (otherObject == null) {
            return false;
        } else if (this.getClass() != otherObject.getClass()) {
            return false;
        } else {
            Player otherPlayer = (Player)otherObject;
            return (this.color.equals(otherPlayer.color)
                    && this.name.equals(otherPlayer.name)
                    && this.colorName.equals(otherPlayer.colorName)
                    );
        }
    }
    public int getArmy() {
```

```java
        return army;
    }
    public void addArmy() {
        this.army++;
    }
    public Component getPieceLabel(int i) {
        return pieceLabels[i];
    }

    //Method to use devCard(type) can only use one at a time
    //remove from array
    //activate item chosen (increase VP, add knight, or progress
card(multiple types))

    //Method to determine largest army
    //if (knights cards in play >= 3) true

    //Method to determine longest road
    // if (roadCount > all other players) true

}
```