```java
package jumpgame;

/**
 * JumpGame.java - the main class for the application
 * @version 1.0
 * @since 2014-01-14
 * @author Jack Gore
 */
public class JumpGame
{

    /**
     * main method for the application
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        StartScreen startScreen = new StartScreen();
        Display.title = "Jump";
    }

}
```

```
package jumpgame;

/**
 * required imports
 */
import javax.swing.JLabel;

/**
 * GameObject.java - represents an object in a game
 * @version 1.0
 * @since 2014-01-14
 * @author Jack Gore
 */
public class GameObject
{

    public  Coordinates coordinates;
    public  JLabel      image;
    private boolean     isAlive;

    /**
     * default constructor for the class
     */
    public GameObject(){
        dispose();
        coordinates = new Coordinates();
        spawn();
    }

    /**
     * constructor for the class
     * @param image the image for this object
     */
    public GameObject(JLabel image){
        dispose();
        coordinates = new Coordinates();
        this.image = image;
        spawn();
    }

    /**
     * disposes of object resources
 */
protected void dispose() {
    isAlive = false;
    if (coordinates != null) coordinates.dispose();
    image = null;
}

/**
 * spawns the object to be alive
 */
protected void spawn(){
    isAlive = true;
    update();
}

/**
 * updates coordinates from the image location
 */
public void update() {
    if (image != null) {
        coordinates.x = image.getX();
        coordinates.y = image.getY();
        coordinates.width = image.getWidth();
        coordinates.height = image.getHeight();
    }
    coordinates.recalculate();
}

/**
 * determines if the object is colliding with another game object
 * vertically
 * @param target the game object to check against
 * @return it is colliding (true) or not (false)
 */
public boolean isCollidingVertivally(GameObject target) {
    if (this.isAlive && target.isAlive){
        if (this.coordinates.top >= target.coordinates.top &&
            this.coordinates.top <= target.coordinates.bottom)
            return true;
        else if (target.coordinates.top >= this.coordinates.top &&
                 target.coordinates.top <= this.coordinates.bottom)
            return true;
        else if (this.coordinates.bottom >= target.coordinates.top
                    this.coordinates.bottom <= target.coordinates.bott
                return true;
            else if (target.coordinates.bottom >= this.coordinates.top
                    target.coordinates.bottom <= this.coordinates.bott
                return true;
        }
        return false;
    }

    /**
     * determines if the object is colliding with another game object
     * horizontally
     * @param target the game object to check against
     * @return it is colliding (true) or not (false)
     */
    public boolean isCollidingHorizontally(GameObject target) {
        if (this.isAlive && target.isAlive){
            if (this.coordinates.left >= target.coordinates.left &&
                this.coordinates.left <= target.coordinates.right)
                return true;
            else if (target.coordinates.left >= this.coordinates.left &
                    target.coordinates.left <= this.coordinates.right)
                return true;
            else if (this.coordinates.right >= target.coordinates.left
                    this.coordinates.right <= target.coordinates.right
                return true;
            else if (target.coordinates.right >= this.coordinates.left
                    target.coordinates.right <= this.coordinates.right
                return true;
        }
        return false;
    }

    /**
     * places the image at the stored coordinates
     */
    public void redraw() {
        image.setBounds(coordinates.x, coordinates.y,
                coordinates.width, coordinates.height);
    }

}
```

```
om)

&&
om)
```

```
&


&&
)

&&
)
```

```java
package jumpgame;

/**
 * required imports
 */
import java.awt.Color;
import java.awt.Container;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;

/**
 * StartScreen.java - the starting GUI for the application
 * @version 1.0
 * @since 2014-01-14
 * @author Jack Gore
 */
public class StartScreen extends JFrame
{

    private JLabel lblJump;
    private JButton btnPlay;
    private JButton btnQuit;
    private Container container;
    public boolean isPlay = false;

    /**
     * default constructor for the class
     */
    public StartScreen() {
        Font labelFont = new Font("Consolas", Font.BOLD, 72);
        final int LABEL_WIDTH = 200;
        final int LABEL_HEIGHT = 100;
        final int BUTTON_WIDTH = 100;
        final int BUTTON_HEIGHT = 25;
        container = this.getContentPane();
        container.setLayout(null);
        lblJump = new JLabel("JUMP");
        btnPlay = new JButton("Play");
    btnQuit = new JButton("Quit");
    container.add(lblJump);
    container.add(btnPlay);
    container.add(btnQuit);
    lblJump.setFont(labelFont);
    lblJump.setForeground(Color.BLUE);
    lblJump.setBounds(165, 75, LABEL_WIDTH, LABEL_HEIGHT);
    btnPlay.setBounds(115, 200, BUTTON_WIDTH, BUTTON_HEIGHT);
    btnPlay.setForeground(Color.red);
    btnQuit.setBounds(300, 200, BUTTON_WIDTH, BUTTON_HEIGHT);
    btnQuit.setForeground(Color.red);
    ActionListener playButton = new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e) {
            play();
        }
    };
    btnPlay.addActionListener(playButton);
    ActionListener quitButton = new ActionListener(){
        @Override
        public void actionPerformed(ActionEvent e) {
            thxPlaying();
            System.exit(0);
        }
    };
    btnQuit.addActionListener(quitButton);
    this.setUndecorated(true);
    this.setTitle("JUMP");
    this.setSize(500, 300);
    this.setVisible(true);
    this.setResizable(false);
    this.setLocationRelativeTo(null);
}

/**
 * ending game message
 */
private void thxPlaying(){
    Display.output("Thanks for coming!\nGood-Bye!");
    System.exit(0);
}

/**
```

```
     * moves the application to the game GUI
     */
    private void play() {
        GameScreen gameScreeen = new GameScreen();
        this.dispose();
    }

}
```

```java
package jumpgame;

/**
 * required imports
 */
import java.awt.Color;
import java.awt.Container;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JFrame;
import javax.swing.JLabel;

/**
 * GameScreen.java - the main game screen of the game
 * @version 1.0
 * @since 2014-01-14
 * @author Jack Gore
 */
public class GameScreen extends JFrame
{

    private final int MAX_WALLS = 3;

    private JLabel      lblSquare;          // GUI labels (images)
    private JLabel[]    lblWalls;
    private JLabel      lblGround;
    private Container   container;          // GUI container
    private Player      player;             // Game objects
    private Wall[]      walls;
    private Wall        ground;
    private KeyListener keyListener;        // Action listener objects

    /**
     * default constructor for the class
     */
    public GameScreen() {
        setUpGUI();
        setGameObjects();
        setKeyboardListening();
        startGame();
    }
```

```
/**
 * keyboard press action
 * @param e the registered key event
 */
private void keyAction(KeyEvent e) {
    int key = e.getKeyCode();
    if      (key == KeyEvent.VK_SPACE) player.jump();
    else if (key == KeyEvent.VK_RIGHT) player.moveRight();
    else if (key == KeyEvent.VK_LEFT)  player.moveLeft();
    else if (key == KeyEvent.VK_Q)     thxPlaying();
}

/**
 * ending game message
 */
private void thxPlaying(){
    Display.output("Thanks for coming!\nGood-Bye!");
    System.exit(0);
}

/**
 * set up the GUI for the game, placing all GUI components
 */
private void setUpGUI() {
    // set container objecgt for GUI components
    container = this.getContentPane();
    container.setLayout(null);
    // instantiate labels (images)
    lblGround = new JLabel("GROUND");
    lblSquare = new JLabel("PLAYER");
    lblWalls = new JLabel[MAX_WALLS];
    for (int i = 0; i < lblWalls.length; i++) {
        lblWalls[i] = new JLabel("WALL " + i);
    }
    // create variables for positioning objects
    final int wallSpacing = 285;
    final int frameWidth = 750;
    final int frameHeight = 500;
    final int wallWidth = 225;
    final int wallHeight = 20;
    final int squareHeight = 42;
    final int squareWidth = 42;
    final int wallStartX = (frameWidth / 2) - (wallWidth / 2);
    final int wallStartY = (frameHeight / 2) - (wallHeight / 2);
```

```java
        final int wallStartY = (frameHeight / 2) - (wallHeight / 2);
        final int wallTwoStartX = (wallStartX + wallSpacing + 50);
        final int wallThreeStartX = (wallTwoStartX + wallSpacing + 50);
        final int groundHeight = 20;
        final int groundWidth = frameWidth;
        final int groundSrtX = 0;
        final int groundSrtY = (frameHeight - groundHeight);
        // add GUI components to container
        container.add(lblGround);
        container.add(lblSquare);
        // set look and feel of walls
        for (int i = 0; i < lblWalls.length; i++) {
            container.add(lblWalls[i]);
            lblWalls[i].setOpaque(true);
            lblWalls[i].setBackground(Color.blue);
            lblWalls[i].setVisible(true);
        }
        lblWalls[0].setBounds(wallStartX, wallStartY, wallWidth, wallHe
        lblWalls[1].setBounds(wallTwoStartX, wallStartY, wallWidth, wal
        lblWalls[2].setBounds(wallThreeStartX, wallStartY, wallWidth, w
        // ground
        lblGround.setOpaque(true);
        lblGround.setBackground(Color.orange);
        lblGround.setVisible(true);
        lblGround.setBounds(groundSrtX, groundSrtY, groundWidth, ground
        // player
        lblSquare.setOpaque(true);
        lblSquare.setBackground(Color.red);
        lblSquare.setVisible(true);
        lblSquare.setBounds(wallStartX, (wallStartY - squareHeight - 1)
        // set look and feel of JFrame
        this.setSize(frameWidth, frameHeight);
        this.setUndecorated(true);
        this.setLocationRelativeTo(null);
    }

    /**
     * sets up the listening event for the keyboard
     */
    private void setKeyboardListening() {
        keyListener = new KeyListener() {
            public void keyTyped(KeyEvent e)    { keyAction(e); }
            public void keyPressed(KeyEvent e)  { keyAction(e); }
            public void keyReleased(KeyEvent e) { keyAction(e); }
```

```
ight);
lHeight);
allHeight);



Height);



, squareWidth, squareHeight);
```

```
            public void keyReleased(KeyEvent e) { keyAction(e); }
        };
        container.addKeyListener(keyListener);
        this.addKeyListener(keyListener);
    }

    /**
     * creates the needed game objects
     */
    private void setGameObjects() {
        ground = new Wall(lblGround,this);
        walls = new Wall[MAX_WALLS];
        for (int i = 0; i < walls.length; i++) {
            walls[i]  = new Wall(lblWalls[i],this);
        }
        player = new Player(lblSquare,walls,ground,this);
    }

    /**
     * starts the game
     */
    private void startGame() {
        player.start();
        for (int i = 0; i < walls.length; i++) {
            walls[i].start();
        }
        this.setVisible(true);
    }

}
```

```
package jumpgame;

/**
 * required imports
 */
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JLabel;
import javax.swing.Timer;

/**
 * Player.java - represents a game player in this game
 * @version 1.0
 * @since 2014-01-14
 * @author Jack Gore
 */
public class Player extends GameObject
{

    // global constants
    private final int PLAYER_TIMER_DELAY  = 20;
    private final int GRAVITY_TIMER_DELAY = 20;
    private final int JUMP_TIMER_DELAY    = 50;
    private final int GRAVITY_MOVE_AMOUNT = 1;
    private final int MOVE_AMOUNT         = 1;
    private final int MAX_JUMP_HEIGHT     = 70;

    private Timer          verticalMoveTimer;
    private Timer          gravityTimer;
    private Timer          jumpTimer;
    private ActionListener verticalMoveListener;
    private ActionListener gravityListener;
    private ActionListener jumpListener;
    private Wall[]         walls;
    private Wall           ground;
    private GameScreen     gameScreen;
    private int            jumpHeight;
    private boolean        isInJump;

    /**
     * constructor for the class
     * @param image the image for this object
 * @param walls the walls for the player
 * @param ground the ground for the player
 * @param gameScreen the game screen GUI
 */
public Player(JLabel image, Wall[] walls, Wall ground,
        GameScreen gameScreen){
    super(image);
    super.coordinates.direction = Directions.STOP;
    super.coordinates.amount    = MOVE_AMOUNT;
    setPlayerTimers();
    this.walls      = walls;
    this.ground     = ground;
    this.gameScreen = gameScreen;
}

/**
 * starts the game
 */
public void start() {
    verticalMoveTimer.start();
    gravityTimer.start();
}

/**
 * sets up the timer objects for the game
 */
private void setPlayerTimers() {
    verticalMoveListener = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            verticalMoveAction();
        }
    };
    gravityListener = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            gravityAction();
        }
    };
    jumpListener = new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            jumpAction();
        }
    };
    verticalMoveTimer  = new Timer(PLAYER_TIMER_DELAY,  verticalMov
eListener);
    `
    gravityTimer          = new Timer(GRAVITY_TIMER_DELAY, gravityList
    jumpTimer             = new Timer(JUMP_TIMER_DELAY,    jumpListene
}

/**
 * the jump action the game player calls for
 */
public void jump() {
    if (isInJump == false) {
        isInJump = true;
        gravityTimer.stop();
        jumpTimer.start();
    }
}

/**
 * the move left action the game player calls for
 */
public void moveRight() {
    coordinates.moveRight();
}

/**
 * the move right action the game player calls for
 */
public void moveLeft() {
    coordinates.moveLeft();
}

/**
 * the vertical movement action for the vertical movement timer
 */
private void verticalMoveAction() {
    update();
    checkVerticalDirection();
    checkSideCollisions();
    redraw();
}

/**
 * the gravity movement action for the gravity timer
 */
private void gravityAction() {
    update();
ener);
r);
```

```
        update();
        moveDown();
        checkDownwardCollisions();
        redraw();
    }

    /**
     * the jump movement action for the jump timer
     */
    private void jumpAction() {
        if (jumpHeight >= MAX_JUMP_HEIGHT) {
            gravityTimer.start();
            jumpTimer.stop();
            jumpHeight = 0;
        }
        else {
            jumpHeight++;
            update();
            coordinates.y = coordinates.y - GRAVITY_MOVE_AMOUNT;
            coordinates.recalculate();
            redraw();
        }
    }

    /**
     * checks the current vertical direction and moves
     */
    private void checkVerticalDirection() {
        if      (coordinates.direction == Directions.LEFT)
            coordinates.moveLeft();
        else if (coordinates.direction == Directions.RIGHT)
            coordinates.moveRight();
        else if (coordinates.direction == Directions.STOP)
            coordinates.direction = Directions.STOP;
    }

    /**
     * checks for collisions against side objects
     */
    private void checkSideCollisions() {
        for (int i = 0; i < walls.length; i++) {
            if (isTouching(walls[i])) {
                stickTo(walls[i]);
            }
            }
        }

        /**
         * checks for collisions while moving down
         */
        private void checkDownwardCollisions() {
            if (isTouching(ground)) {
                verticalMoveTimer.stop();
                gravityTimer.stop();
                Display.output("You hit the ground!");
            }
            for (int i = 0; i < walls.length; i++) {
                if (isTouching(walls[i])) {
                    landOn(walls[i]);
                    i = walls.length;
                }
            }
        }

        /**
         * determines if the object is touching a wall
         * @param wall the wall object to check
         * @return is touching (true) or not (false)
         */
        private boolean isTouching(Wall wall) {
            if (isCollidingHorizontally(wall) &&
                isCollidingVertivally(wall)) {
                return true;
            }
            return false;
        }

        /**
         * sticks the coordinates to the wall object
         * @param wall the wall object to stick to
         */
        private void stickTo(Wall wall) {
            if (coordinates.direction == Directions.LEFT)
                coordinates.x = wall.coordinates.right + 1;
            else if (coordinates.direction == Directions.RIGHT)
                coordinates.x = wall.coordinates.left - 1;
            coordinates.recalculate();
```

```java
    }

    /**
     * lands on the current wall object
     * @param wall the wall object to land on
     */
    private void landOn(Wall wall) {
        coordinates.y = wall.coordinates.top - coordinates.height - 1;
        coordinates.recalculate();
        isInJump = false;
    }

    /**
     * moves the objects down
     */
    private void moveDown() {
        coordinates.y = coordinates.y + GRAVITY_MOVE_AMOUNT;
        coordinates.recalculate();
    }

}
```

```
package jumpgame;

/**
 * required imports
 */
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JLabel;
import javax.swing.Timer;

/**
 * Wall.java - represents a wall in this game
 * @version 1.0
 * @since 2014-01-14
 * @author Jack Gore
 */
public class Wall extends GameObject
{

    private final int WALL_TIMER_DELAY = 50;
    private final int WALL_MOVE_AMOUNT = 1;

    private Timer          wallTimer;
    private ActionListener wallListener;
    private GameScreen     gameScreen;

    /**
     * constructor for the class
     * @param image
     * @param gameScreen
     */
    public Wall(JLabel image, GameScreen gameScreen) {
        super(image);
        wallListener = new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                wallAction();
            }
        };
        wallTimer = new Timer(WALL_TIMER_DELAY,wallListener);
        this.gameScreen       = gameScreen;
        coordinates.direction = Directions.LEFT;
        coordinates.amount    = WALL_MOVE_AMOUNT;
    }

    /**
     * starts the wall moving
     */
    public void start() {
        wallTimer.start();
    }

    /**
     * the timer action for movement
     */
    private void wallAction() {
        update();
        coordinates.moveLeft();
        checkCollisions();
        redraw();
    }

    /**
     * checks the wall to see if it collides with the screen edge
     */
    private void checkCollisions() {
        if (coordinates.right < 0) {
            coordinates.x = gameScreen.getWidth();
            coordinates.recalculate();
        }
    }

}
```

```java
package jumpgame;

/**
 * required imports
 */
import javax.swing.JFrame;
import javax.swing.JOptionPane;

/**
 * Display.java - used for displaying messages
 * @version 1.0
 * @since 2014-01-14
 * @author Jack Gore
 */
public class Display
{

    public static String title = "";
    public static JFrame gui   = null;

    /**
     * outputs the text in a dialog message box
     * @param text the text to display
     */
    public static void output(String text) {
        JOptionPane.showMessageDialog(gui,text,title,
                JOptionPane.PLAIN_MESSAGE,null);
    }

    /**
     * outputs the text to the system output
     * @param text the text to display
     */
    static void print(String text) {
        System.out.println(text);
    }

}
```

```java
package jumpgame;

/**
 * Coordinates.java - stores the coordinates of an object
 * @version 1.0
 * @since 2014-01-14
 * @author Jack Gore
 */
public class Coordinates
{

    public int x, y, left, right, top, bottom,
               width, height, direction, amount;

    /**
     * default constructor for the class
     */
    public Coordinates() {
        dispose();
    }

    /**
     * disposes of object resources
     */
    public void dispose() {
        x = y = left = right = top = bottom =
            width = height = direction = amount = 0;
    }

    /**
     * moves the coordinates up
     */
    public void moveUp(){
        direction = Directions.DOWN;
        y = y - amount;
        recalculate();
    }

    /**
     * moves the coordinates down
     */
    public void moveDown(){
        direction = Directions.DOWN;
        y = y + amount;
        recalculate();
    }

    /**
     * moves the coordinates left
     */
    public void moveLeft(){
        direction = Directions.LEFT;
        x = x - amount;
        recalculate();
    }

    /**
     * moves the coordinates right
     */
    public void moveRight(){
        direction = Directions.RIGHT;
        x = x + amount;
        recalculate();
    }

    /**
     * moves the coordinates north
     */
    public void moveNorth(){
        moveUp();
    }

    /**
     * moves the coordinates south
     */
    public void moveSouth(){
        moveDown();
    }

    /**
     * moves the coordinates east
     */
    public void moveEast(){
        moveRight();
    }
    /**
     * moves the coordinates west
     */
    public void moveWest(){
        moveLeft();
    }

    /**
     * recalculates all other coordinates
     */
    public void recalculate() {
        left = x;
        top = y;
        right = left + width;
        bottom = top + height;
    }

}
```

```java
package jumpgame;

/**
 * Directions.java - stores direction values
 * @version 1.0
 * @since 2014-01-14
 * @author Jack Gore
 */
public class Directions
{

    public static final int STOP        = 0;
    public static final int NORTH       = 1;
    public static final int WEST        = 2;
    public static final int SOUTH       = 3;
    public static final int EAST        = 4;

    public static final int UP      = NORTH;
    public static final int DOWN    = SOUTH;
    public static final int LEFT    = WEST;
    public static final int RIGHT   = EAST;

}
```