MAY 12, 2019

Computer Graphics

CSE3323

Prof. Ali ElDesoky

GRAPHICS PROJECTS

CREATING GRAPHICS PROJECTS USING JAVA APPLETS

ABDULRAHMAN ISSAM ABOUOUF
FACULTY OF ENGINEERING, MANSOURA UNIVERSITY
CSED 2020

Contents

Table Of Figures

1 Ping Pong Game

This Project is a game of multiplayer that simulates a Ping Pong game between two players

The main objects are the Paddle and the Ball. The following will show how they are created in depth.

1.1 Paddle Interface

The first thing created is the player paddle, first we will create the interface to be used in the HumanPaddle class and (optional) AIPaddle.

```
package PingPong1;
import java.awt.*;

public interface Paddle {
	public void draw(Graphics g);
	public void move();
	public int getY();


}
```

1.2 Player Paddle

The second step is to create the HummanPaddle class to create objects from it in our applet. First we will use Up and Down Accel variables to determine the direction if up or down of the paddle and the constructor of the class as to create which paddle if for player one or player two as follows.

```
	double y,yVel;
	boolean upAccel,downAccel;
	int player,x; //x and y represent the place of the paddle and player spicefies wether
it's player on or two
	final double GRAVITY = 0.94;

	public HumanPaddle(int player) { //the constructor is called to know which paddle of
each player
		upAccel=false; downAccel=false;
		y = 210; yVel = 0;
		if(player==1) x=20;
		else x=660;
	}
```

1.2.1 Moving the paddle

To move the paddle we will run this move function in an infinite loop in our run method in the applet as follows , the move function is created in the HumanPaddle Class

```
public void move() {
		if(upAccel) {
			yVel -=2; //for the paddle to go up
		}
		else if(downAccel) {
			yVel +=2; //for the paddle to go down
		}
		else if(!upAccel && !downAccel) {
			yVel*= GRAVITY; //For Accelration
		}


		if(yVel>=5) yVel = 5;
		else if(yVel <=-5) yVel = -5;

		y += yVel;
		if(y<=0) y=0;
		if(y>=420) y=420;


	}
```

1.2.2 Setting the positions of the paddle depending on the keys pressed

```
	public void setUpAccel(boolean input) {
		upAccel = input;
	}

	public void setDownAccel(boolean input) {
		downAccel = input;
	}

	@Override
	public int getY() {
		return (int)y;

	}
```

1.2.3 Drawing the paddle

This function is called in the paint(Graphics g) to paint the player paddle as follows, this function is in HumanPaddle Class

```
public void draw(Graphics g) {
		g.setColor(Color.red);
		g.fillRect(x, (int)y, 20, 80);


	}
```

1.3 The ball

Creating a ball class to handle the collisions of the paddle and the object of the ball

1.3.1 Handling the collision

In the Ball class the collision function will get the two paddles of the players and handle the hitting of the ball to reflect it back in the game as follows.

```
public void checkPaddleCollision(Paddle p1, Paddle p2) {
            if(x<=50) { //the first player
                  if(y >= p1.getY() && y<= p1.getY() + 80) {
// this means as the paddle is 80 in height that the ball
touch
                        xVel = -xVel;
                        clip.play();
                  }
            }

            else if (x>= 650) { //the second player
                  if(y >= p2.getY() && y<= p2.getY() + 80) {
                        xVel = -xVel;
                        clip.play();
                  }
            }

      }
```

1.3.2 Moving The ball after the hit

For the ball to stay in border and also to get out if the paddle didn't hit it this function will run as follows

```
public void move() {
             x+=xVel;
             y+=yVel;

             if(y<10) yVel = -yVel;
             if(y>490) yVel = -yVel;

}
```

1.3.3 Drawing the ball

This draw function will be called in the applet paint

```
public void draw(Graphics g) {
            g.setColor(Color.orange);
            g.fillOval((int)x-10,(int) y-10, 20, 20);

}
```

1.4 Running the game and setting the applet environment

Our Tennis class is the applet class that will call all the above classes and handle the game as follows, it uses Runnable to run the game and KeyListner to read from the keyboard

The controllers are (W,D) for player one and (O,L) for player two.

It may use the double buffering GFX to create an image of the game to handle the game flicking.

```
public class Tennis extends Applet implements Runnable, KeyListener{
	final int WIDTH = 700, HEIGHT=500;
	HumanPaddle p1;
	HumanPaddle p2;
	Ball b1;
	Font f1;
	Graphics gfx;
	Image img;
	Thread thread;
	Label testLabel;

	//private static final long serialVersionUID = 2530894095587089544L;
    //private AudioClip clip;

	public void init() {
		this.resize(WIDTH, HEIGHT);
		this.setFocusable(true);
		this.addKeyListener(this);
		f1 = new Font("Arial",Font.BOLD,30);
	    //clip = getAudioClip(getDocumentBase(), "sound.wave");

		p1 = new HumanPaddle(1); //makes a Paddle in the Left from the created
class
		p2 = new HumanPaddle(2);
		b1 = new Ball();
		img = createImage(WIDTH,HEIGHT);
		gfx = img.getGraphics();
		thread = new Thread(this);
		thread.start();

	}

	public void paint (Graphics g) {
		this.setBackground(Color.blue);
		g.setColor(Color.white);
		g.fillRect(0, 250, WIDTH, 20);
		g.setColor(Color.DARK_GRAY); //setting the ping pong table
		g.fillRect(350, 0, 5, HEIGHT);
		super.paint(g);

		if(b1.getX()<-10 || b1.getX()> 710) {
			g.setColor(Color.red);//after the game is over
			g.setFont(f1);
			g.drawString("Game Over", 260, 250);
		}
```

```
			else {
				p1.draw(g);
				p2.draw(g);
				b1.draw(g);
			}

		}

//		public void update(Graphics g) {
//			//paint(g);
//		}

		@Override
		public void run() {
			// TODO Auto-generated method stub
			for(;;) {

				p1.move();
				p2.move();
				b1.move();
				b1.checkPaddleCollision(p1, p2); // check collision

				repaint(); //to run the game over and over
				try {
					Thread.sleep(10);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}

			}
		}

		@Override
		public void keyPressed(KeyEvent e) {
			if(e.getKeyCode() == KeyEvent.VK_W) {
				p1.setUpAccel(true);
			}
			else if(e.getKeyCode() == KeyEvent.VK_S) {
				p1.setDownAccel(true);
			}

			if(e.getKeyCode() == KeyEvent.VK_O) {
				p2.setUpAccel(true);
			}
			else if(e.getKeyCode() == KeyEvent.VK_L) {
				p2.setDownAccel(true);
			}


		}

		@Override
		public void keyReleased(KeyEvent e) {
			if(e.getKeyCode() == KeyEvent.VK_W) {
				p1.setUpAccel(false);
```

```
                }
                else if(e.getKeyCode() == KeyEvent.VK_S) {
                        p1.setDownAccel(false);
                }

                if(e.getKeyCode() == KeyEvent.VK_O) {
                        p2.setUpAccel(false);
                }
                else if(e.getKeyCode() == KeyEvent.VK_L) {
                        p2.setDownAccel(false);
                }
        }

}
```

1.5 The Result

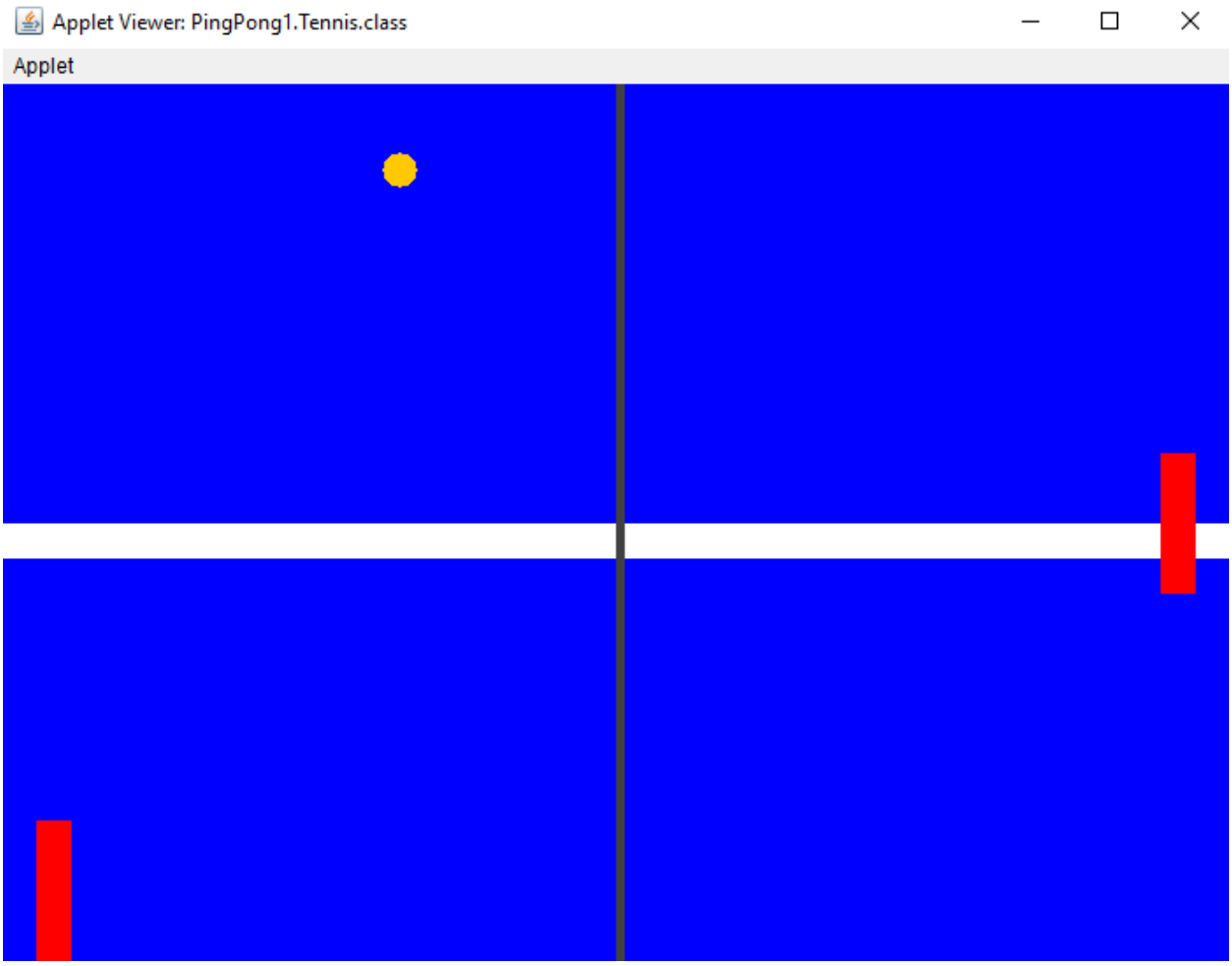

Figure 1-1 Ping Pong Game Applet

2 Snake Game

This is the traditional game of Snake that eats tokens and the player role is to keep eating random tokens without getting out of border and also without getting out of the board.

To create this game isn't as easy as it sounds as it uses randoms, collision equations and Lists of Points.

The next few pages will show in depth how the game is created.

2.1 Points

Think of it as the snake you are controlling is a list of points of x and y and they are connected and also the tokens are a random point that shows in the board game

To handle this, we will need a Point Class to get the point and also get set the point as we move, also to create a list of points for the snake.

```
package Snake;

public class Point {
	private int x,y;

	public Point() {
		x=0;
		y=0;
	}

	public Point (int x, int y) {
		this.x = x;
		this.y = y;
	}

	public void setX(int x) {
		this.x = x;
	}

	public void setY(int y) {
		this.y = y;
	}

	public int getX() {
		return x;
	}
	public int getY() {
		return y;
	}


}
```

2.2 Creating A Snake Class

Object of a Snake will be created during the game. This sets the snake first, handles the collisions and snake collisions and controlling the movement of the snake.

2.2.1 Setting the initial snake and Drawing the snake

This sets a snake of 20 points connected in the List and when it is drawn each point will be a rectangle as follows

```
List<Point> snakePoints;
	int xDir, yDir;
	boolean isMoving; //to know if the snake is moving
	boolean elongate; //to know when to make taller each time it eats
	final int STARTSIZE = 20, STARTX = 150, STARTY = 150; //initial condition
of the snake
	public Snake() {
		snakePoints = new ArrayList<Point>();
		xDir = 0;
		yDir = 0;
		isMoving = false;
		elongate = false;
		snakePoints.add(new Point(STARTX, STARTY));

		for(int i=1;i<STARTSIZE; i++) {
			snakePoints.add(new Point(STARTX-i*4, STARTY)); //add a point
to the body of the snake to the left and as each point is assigned as 4 sizerect

		}
	}
```

```
public void draw(Graphics g) {
		g.setColor(Color.white);
		for (Point p : snakePoints) {
			g.fillRect(p.getX(), p.getY(), 4, 4);
		}
	}
```

2.2.2 Moving the snake

Moving the snake is a tedious task, first we will get the head of the snake, this is the first element of the snakePoints and the newStart as every head moving will be changed depending on the directions from the player (**xDir and yDir**) **and the *4 is used as each point is a rectangle of 4** then setting each point to the next point of the snake, as –think of it- when the snake move the points are just shifted to the next points and the loop continues.

Here is the code with the explanation comments,

```
public void move() {
            if (isMoving) { //boolean as in first the snake is paused but when the player hits a key
this will be true
                  Point temphead = snakePoints.get(0);
                  Point temptail = snakePoints.get(snakePoints.size()-1);
                  Point newStart = new Point(temphead.getX() + xDir*4 , temphead.getY() + yDir*4);

                  for(int i = snakePoints.size() - 1; i>=1; i--) {
                        snakePoints.set(i, snakePoints.get(i-1));
                  }
                  snakePoints.set(0, newStart);
                  if(elongate) {
                        for(int i=0; i<6; i++) snakePoints.add(temptail);
                        elongate = false;
                  }
            }

      }
```

2.2.3 Snake Collision

What if the snake hits itself? The game is over this code gets the head of the snake (getX() and getY()) and checks if it equals any point of the snakePoints if true this means a collision.

```
public boolean snakeCollision() { //if snake collides with itself
            int x = this.getX();
            int y = this.getY();
            for(int i = 1; i<snakePoints.size(); i++) { // getX and getY get the
head of the snake and if any point except the head equals this means a bam!
collision
                  if(snakePoints.get(i).getX() == x &&
snakePoints.get(i).getY()==y) {
                        return true:
                  }
            }
            return false;
      }
```

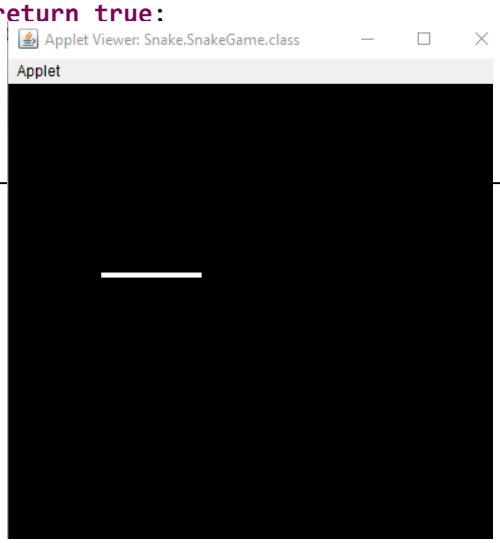

Figure 2-1 Initalized Snake

2.3 The Token (the eatable piece)

This token is generated randomly in the board and random functions generates (0 → 1) multiplying by the border width and height. The token takes an object of a snake to get the parameters of it to check collisions between them etc…

It also keeps the score of how many tokens the snake ate and makes the snake taller each time.

2.3.1 Randomizing the token place

```
private int x,y,score;
	private Snake snake;

	public Token(Snake s) {
		x = (int)(Math.random()*395);
		y = (int)(Math.random()*395);
		snake =  s;
	}
```

2.3.2 Snake and Token Collision

This function checks if a collision happens, if true the function will call to generate another random token and also increase the score.

```
public boolean tokenCollision() {//if the snake hits the token
		int snakeX = snake.getX() +2; //get the snake head position ,  the 2 added is to get the midde of the head
		int snakeY = snake.getY()+2;

		if(snakeX>=x-1 && snakeX<=(x+7)) { //since the token is 6 in size the 7 and 1 are used so any touched bit of the token
			if(snakeY>=y-1 && snakeY<=(y+7)) {
				changePosition(); //if the snake hits the token, randomize another token function
				snake.setElongte(true);
				score++;
				return true;
			}
		}
		return false;

	}
```

2.3.3 Getting the Score and drawing the token

In the main game applet these will be called.

2.4 Running the Applet and Seeing the environment

This is the main class of the game that will set the board, call the classes above, determining the game sequence, getting the controls from the user (W,A,S,D) and checks if the game is over or not.

Here is its full code with explanation comments

```
public class SnakeGame extends Applet implements Runnable, KeyListener{
	Graphics gfx;
	Image img;
	Snake snake; //create a snake
	Token token; //creates a
token
	Font f1,f2;
	boolean gameOver;
	Thread thread;
	public void init() {

	this.setFocusable(true);

	this.addKeyListener(this);
		this.resize(400,
400);
		f1 = new Font("Arial",Font.BOLD,30);
	    f2 = new Font("Forte",Font.PLAIN,26);
		this.setBackground(Color.black);
		snake = new Snake();
		token = new Token(snake);
		img = createImage(400,400);
		gfx = img.getGraphics();
		thread = new Thread(this); //for the runnable
		thread.start();
	}

	public void paint(Graphics g) {
		if(!gameOver) {
			snake.draw(g);
			token.draw(g);
		}
		else {
			g.setColor(Color.red);
			g.setFont(f1);
			g.drawString("Game Over", 120, 150);
			g.setFont(f2);
			g.drawString("Score : " + token.getScore(), 150, 170);
		}
```

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

	public void draw(Graphics g) {
		g.setColor(Color.green);
		g.fillOval(x, y, 6, 6);
	}
```

```
			//g.drawImage(img,0,0,null);
		}

//		public void repaint (Graphics g) {
//			paint(g);
//		}
//
//		public void update (Graphics g) {
//			paint(g);
//		}


		public void run() {
			for(;;) {
				if(!gameOver) {
					snake.move();
					token.tokenCollision(); //set a random token and moved if
the snake eats it
					this.checkGameOver();
				}
				snake.move();
				this.checkGameOver();
				repaint();
				try {
					Thread.sleep(40);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		}

		public void checkGameOver() {
			if(snake.getX()<0 || snake.getX()>396) {
				gameOver = true;
			}

			if(snake.getY()<0 || snake.getY()>396) {
				gameOver = true;
			}

			if(snake.snakeCollision()) gameOver=true;

		}



		public void keyPressed(KeyEvent e) {
			if(!snake.isMoving()) {
				if(e.getKeyCode()== KeyEvent.VK_W ||e.getKeyCode()==
KeyEvent.VK_S || e.getKeyCode()== KeyEvent.VK_D) {
					snake.setIsMoving(true);
				}
			}
```

```
            if(e.getKeyCode()== KeyEvent.VK_W) {
                  if(snake.getYdir() !=1) { //if not going down
                        snake.setYdir(-1); // go up so the snake won't collapse on
itself
                        snake.setXdir(0); // don't move left or right
                  }

            }

            if(e.getKeyCode()== KeyEvent.VK_S) {
                  if(snake.getYdir() !=-1) { //if not going up
                        snake.setYdir(1); // go down
                        snake.setXdir(0); // don't move left or right
                  }

            }
            if(e.getKeyCode()== KeyEvent.VK_A) {
                  if(snake.getXdir() !=1) { //if not going right
                        snake.setXdir(-1); // go left
                        snake.setYdir(0); // don't move up or down
                  }

            }

            if(e.getKeyCode()== KeyEvent.VK_D) {
                  if(snake.getXdir() !=-1) { //if not going left
                        snake.setXdir(1); // go right
                        snake.setYdir(0); // don't move up or down
                  }

            }


      }


      public void keyReleased(KeyEvent arg0) {


      }


      public void keyTyped(KeyEvent arg0) {


      }

}
```

2.5 Result

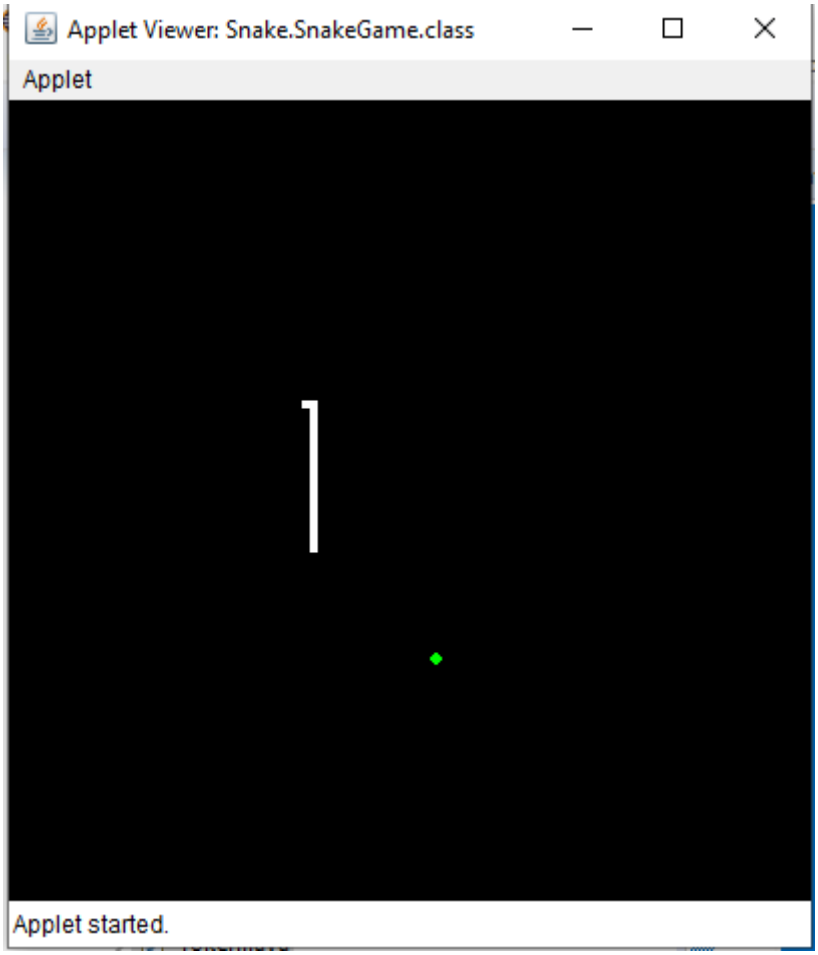

Figure 2-2 Snake Game Applet

3 Drawing Shapes using only drawline()

```
import java.awt.*;

import javax.swing.JOptionPane;

import java.applet.*;

public class chapter extends Applet {
	public int noside, roundangle;
	public void init() {
		noside =
Integer.valueOf(JOptionPane.showInputDialog("enter
number of sides"));
	}
	public void paint(Graphics g) {
		int r=200, c1=300, c2=300;
		roundangle = (360/noside)/2;
		for(int i=0; i<5; i++) {
			Grid(g,c1,c2,r,noside,0);
			Grid(g,c1,c2,r,noside,roundangle);

		}
	}

	public void Grid(Graphics g, int c1, int c2, int r, int noside, int
roundangle) {
		int x1, x2=0, y1, y2=0;
		x1 = (int)(c1 + r*Math.cos(roundangle*3.14/180));
		y1 = (int)(c2 - r*Math.sin(roundangle*3.14/180));

		for(int i= roundangle; i<=360+roundangle; i+=(360/noside)) {
			x1 = (int)(c1 + r*Math.cos(i*3.14/180));
			y1 = (int)(c2 - r*Math.sin(i*3.14/180));
			g.drawLine(x1, y1, x2, y2);
			x2 = x1;
			y2 = y1;

		}
	}

}
```

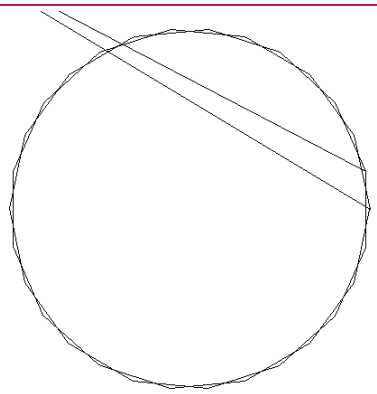

Figure 3-1 15 Shape

4 Drawing Mario with background

```
import javax.swing.JApplet;
import java.awt.*;
import java.util.Random;

public class Mario extends JApplet {
	public void paint(Graphics page) {
		final int MID = 90;  // A line of reference referring to the
	   // imaginary vertical line that runs down
	   // the middle of Mario

		final int TOP = 94;  // Refers to the top of Mario's head (scalp)
	   // as a point of reference
		int count;
	   int layer;
	   int AppletHeight = 203;
	   int AppletWidth = 300;
	  //*******************************************************
	   // Sky
	   //*******************************************************


	   Color sky = new Color (184, 100, 255); // periwinkle-blue

	   page.setColor (sky);
	   page.fillRect (0, 0, AppletWidth, AppletHeight);  // sky

	   Color cloud1 = new Color (230, 140, 255);

	   page.setColor (cloud1);
	   page.fillOval (30, 25, 43, 12); // leftmost

	   Color cloud2 = new Color (240, 185, 255);

	   page.setColor (cloud2);         // overlaps leftmost cloud
	   page.fillOval (53, 14, 77, 16);

	   Color clothes = new Color (225, 38, 38); // red

	  //*******************************************
	   // Super Mario
	   //*******************************************

	   page.setColor (clothes);
	   page.fillOval (MID-9, TOP+17, 18, 24);   // body

	   Color brown = new Color (100, 40, 0);    // dark brown

	   page.setColor (brown);
	   page.fillOval (MID-6, TOP+18, 11, 12);   // shirt
	   page.fillRect (MID-5, TOP+44, 14, 2);    // shoe sole
	   page.fillRect (MID-6, TOP+42, 14, 3);    // shoe lower-body
	   page.fillRect (MID-6, TOP+41, 13, 4);    // shoe upper-body
	   page.fillRect (MID-5, TOP+40, 10, 5);    // shoe peak
```

```
        Color arm = new Color (128, 76, 0);       // lighter version
                                                  // of other brown
                                                  // used on Mario,
                                                  // slight contrast
                                                  // with shirt

        page.setColor (arm);
        page.fillOval (MID-7, TOP+23, 11, 8);     // arm

        Color skin = new Color (255, 190,20);     // Mario's skin color

        page.setColor (skin);
        page.fillOval (MID+4, TOP+24, 5, 6);      // hand
        page.fillOval (MID-10, TOP, 20, 20);      // head

        page.setColor (Color.black);
        page.fillRect (MID+6, TOP+6, 2, 4);       // eye

        page.setColor (skin);
        page.fillOval (MID+9, TOP+8, 5, 6);       // nose

        page.setColor (brown);
        page.fillRect (MID+2, TOP+14, 9, 2);      // mustache
        page.fillRect (MID+4, TOP+12, 2, 4);      // mustache curl

        page.fillRect (MID-10, TOP, 9, 19);       // hair
        page.fillRect (MID-12, TOP+5, 11, 11);    // back of hair

        page.setColor (clothes);
        page.fillRect (MID-10, TOP-1, 19, 5);     // hat base
        page.fillRect (MID-10, TOP+2, 21, 2);     // upper visor
        page.fillRect (MID-10, TOP+3, 23, 1);     // lower visor
        page.fillRect (MID-7, TOP-3, 15, 7);      // hat crown
        page.fillRect (MID-3, TOP-5, 9, 8);       // hat top

        page.setColor (skin);
        page.fillRect (MID-7, TOP+8, 3, 6);       // ear
        page.fillRect (MID-5, TOP+12, 4, 7);      // jaw below
ear


        page.setColor (brown);
        page.fillRect (MID-4, TOP+11, 5, 2);      // sideburn
        page.fillRect (MID-4, TOP+4, 5, 4);       // top hairline

    }

}
```

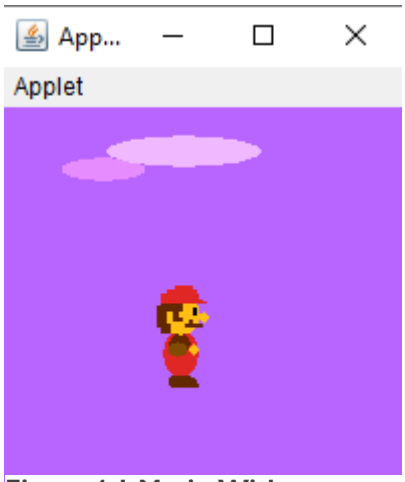

Figure 4-1 Mario With Background

5 Girl Face

```
package chapter6;
import java.awt.*;
import javax.swing.JOptionPane;
import java.applet.*;

public class pg42 extends Applet {
	public void paint(Graphics g) {
		g.fillArc(100, 100, 80, 80, 0, 180); //hair
		g.drawOval(110, 110, 60, 60); //face
		g.setColor(Color.yellow);
		g.fillOval(110, 110, 60, 60); //fill the head yellow
		g.setColor(Color.black);
		g.drawOval(120, 120, 10, 20); //eye
		g.drawOval(150, 120, 10, 20); //eye
		g.fillOval(120, 130, 10, 10);
		g.fillOval(150, 130, 10, 10);
		g.fillOval(100, 140, 10, 10);
		g.fillOval(100, 150, 10, 10);
		g.fillOval(100, 160, 10, 10);
		g.fillOval(170, 140, 10, 10);
		g.fillOval(170, 150, 10, 10);
		g.fillOval(170, 160, 10, 10);
		g.fillArc(130, 150, 20, 20, 180, 0);
	}

}
```

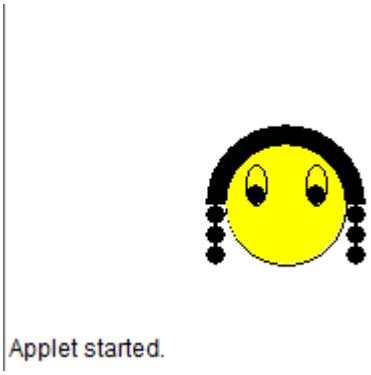

Figure 5-1 Girl Face

6 Train Animation

```
import java.awt.*;
import javax.swing.JOptionPane;
import java.applet.*;

public class pg83 extends Applet {
	int x=0, i=0;

	public void paint(Graphics g) {
		int w = this.getWidth();
		this.setBackground(Color.cyan);
		g.setColor(Color.DARK_GRAY);
		int [] xpoints = {150+x, 250+x, 250+x, 260+x, 260+x, 300+x, 300+x,
290+x,290+x, 260+x, 260+x, 250+x, 250+x, 170+x, 170+x, 150+x};
		int [] ypoints =
{40,40,20,20,40,40,150,150,130,130,150,150,80,80,150,150};
		g.fillPolygon(xpoints, ypoints, 16); //first car
		g.fillRect(x, 40, 140, 90); //Second Car
		g.setColor(Color.black);
		g.fillOval(260+x, 130, 30, 30);
		g.fillOval(170+x, 80, 80, 80); //big wheel
		g.fillOval(10+x, 130, 30, 30);
		g.fillOval(90+x, 130, 30, 30);

		g.setColor(Color.white);
		for(int j=0; j<=360; j+=90) {
			g.fillArc(170+x, 80, 80, 80, j-i, 20);
			g.fillArc(10+x, 130, 30, 30, j-i, 20);
			g.fillArc(90+x, 130, 30, 30, j-i, 20);
			g.fillArc(260+x, 130, 30, 30, j-i, 20);
		}

		g.setColor(Color.GRAY);
		g.fillRect(10+x, 50, 20, 20);
		g.fillRect(60+x, 50, 20, 20);
		g.fillRect(110+x, 50, 20, 20);
		g.fillRect(270+x, 50, 20, 20);

		g.fillRect(140+x, 50, 10, 10);
		g.fillRect(140+x, 110, 10, 10);

		g.setColor(Color.yellow);
		g.fillArc(290+x, 50, 20, 20, 270, 180);

		g.setColor(Color.GRAY);
		g.fillRect(0, 160, w, 100);

		x++;
		if(x==w) x=0;
		i++;
		if(i==360) i=0;
		try {
		Thread.currentThread().sleep(10);
		}
```

```
            catch(Exception ex) {}

            repaint(1);

        }

}
```

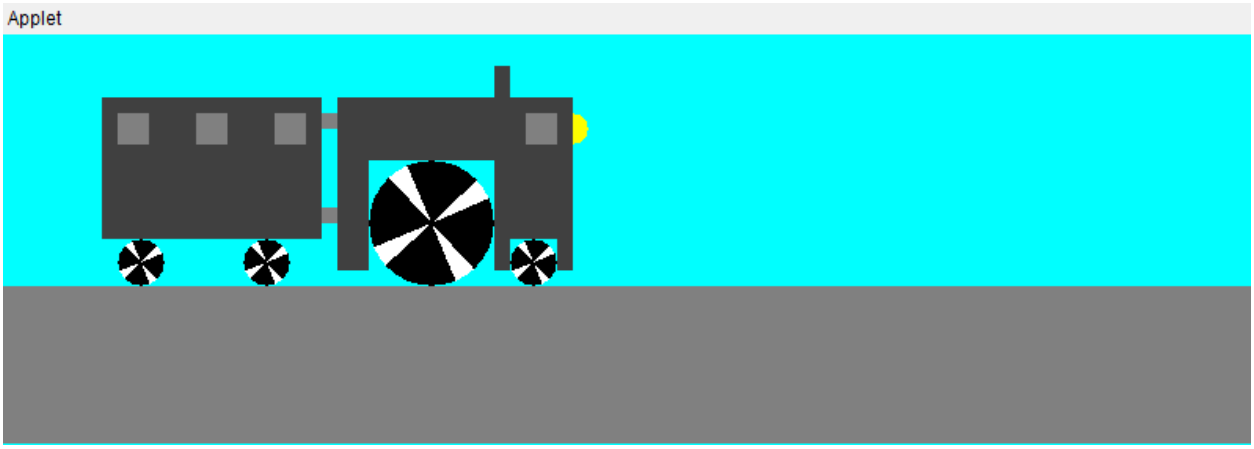

Figure 6-1 Animated Train

7 A Player Hits the Ball Animation

```
import java.awt.*;
import javax.swing.JOptionPane;
import java.applet.*;

public class pg93 extends Applet {
	public void init() {
		this.resize(800, 900);
	}

	int x=0, i=0, j=0, motion = 0; //for the player
	int m=0,n=0; //for the ball
	public void paint (Graphics g) {
		this.setBackground(Color.green);
		g.setColor(Color.white);
		g.fillOval(210+m, 320, 60, 60);
		g.setColor(Color.gray);
		for(int l=0; l<=360; l+=90) {
			g.fillArc(210+m, 320, 60, 60, l+n, 30);
		}

		g.setColor(Color.yellow);
		g.fillOval(10+x, 100, 40, 40);

		int [] x_bhand = {20+x, 40+x, 40+x-j, 80+x-j, 80+x-j, 40+x-j, 20+x-j};
		int [] y_bhand = {160,160,190,190,210,210,190};
		g.fillPolygon(x_bhand, y_bhand, 7);

		g.setColor(Color.red);
		int [] x_body = {x,20+x,20+x,40+x,40+x,60+x,60+x,x};
		int [] y_body = {160,140,130,130,140,150,250,250};
		g.fillPolygon(x_body,y_body, 8);

		g.setColor(Color.yellow);
		g.fillOval(10+x, 100, 40, 40);
		int [] x_fhand = {20+x, 40+x, 40+x+j, 80+x+j, 80+x+j, 40+x+j, 20+x+j};
		int [] y_fhand = {160,160,190,190,210,210,190};
		g.fillPolygon(x_fhand, y_fhand, 7);

		g.setColor(Color.black);
		int [] x_bleg = {x, 60+x, 60+x-j, 80+x-j, 80+x-j, 40+x-j};
		int [] y_bleg = {250,250,360,360,380,380};
		g.fillPolygon(x_bleg, y_bleg,6);

		int [] x_fleg = {x, 60+x, 60+x+j, 80+x+j, 80+x+j, 40+x+j};
		int [] y_fleg = {250,250,360,360,380,380};
		g.fillPolygon(x_fleg, y_fleg,6);


		if(motion==0) {
			if(i==0) {
				i++;
				if(j==20) i=1;
			}
```

```
                if(i==1) {
                        j--;
                        if(j==-20){i=0;
                }
                        x++;
                        if(x==70) motion =1;
        }
    }
        if(motion==1) {
                m++;
                n++;
                if(n==360) n=0;
                if(m==this.getWidth()) m=0;
        }
        try {
                Thread.currentThread().sleep(10);
                }
                catch(Exception ex) {}

                repaint(1);

}
}
```

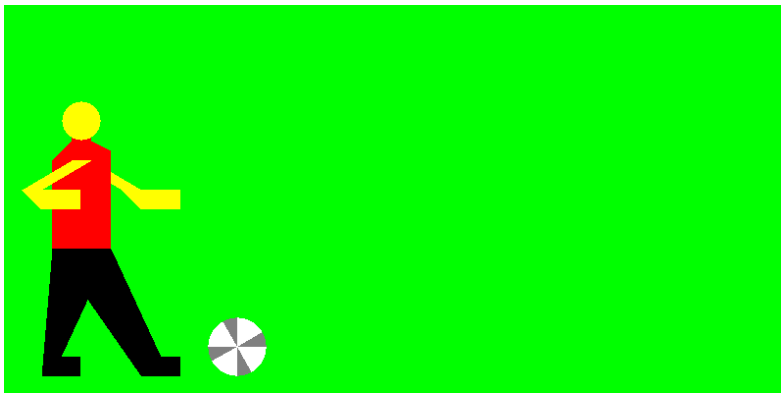

Figure 7-1 Player Before hitting the ball

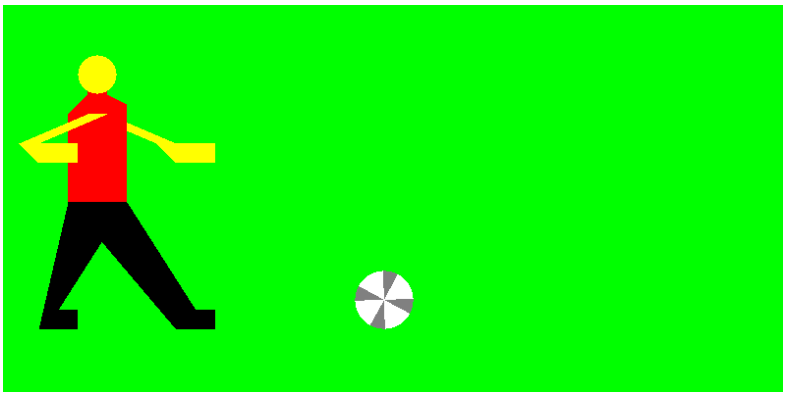

Figure 7-2 Player After hitting the ball

8 Mickey Mouse

```
public void paint (Graphics j){
              j.setColor(Color.black);
              j.fillOval(60+v,50,50,50);
              j.fillOval(170+v,50,50,50);
              j.fillOval(90+v,70,100,90);
              j.setColor(flesh);

              j.fillOval(110+v,80,60,80);
              j.fillArc(90+v,70,100,90,180,180);
              j.fillOval(90+v,110,30,8);
              j.fillOval(160+v,110,30,8);
              j.setColor(Color.black);
              int[] x={130+v,150+v,140+v,130+v};
              int y[]={80,80,90,80};

              int n=4;
              j.fillPolygon(x,y,n);
              j.setColor(Color.white);
              j.fillArc(120+v,95,18,40,0,180);
              j.fillArc(140+v,95,18,40,0,180);
              j.setColor(Color.black);

              j.fillRoundRect(130+v,115,20,10,50,50);
              j.fillOval(125+v,100,10,15);
              j.fillOval(145+v,100,10,15);
              j.drawArc(115+v,120,50,20,190,170);
              j.drawArc(113+v,127,15,20,95,85);
              j.drawArc(153+v,127,15,20,85,-85);

              j.fillArc(130+v,130,30,20,180,200);
              j.setColor(orange);
              j.fillArc(138+v,145,15,10,10,165);
              j.setColor(Color.black);
            int
x1[]={120+v,170+v,210+v,200+v,190+v,170+v,170+v,110+v,110+v,90+v,70+v,120+v};
              int y1[]={160,160,220,230,220,190,220,220,210,210,190,160};

                int n1=12;j.fillPolygon(x1,y1,n1);
              j.setColor(Color.white);
              j.fillOval(70+v,210,50,40);
              j.setColor(Color.RED);

              j.fillArc(38+v,190,133,60,0,90);
              j.fillArc(98+v,180,40,80,135,135);
              j.fillRect(118+v,220,52,39);

              j.setColor(Color.white);
              j.fillOval(160+v,220,50,40);

              j.setColor(Color.black);
              j.drawArc(130+v,218,20,40,180,88);
              int x2[]={118+v,138+v,148+v,128+v,118+v};
              int y2[]={259,259,295,290,259};
```

```
        int n2=5;

        j.fillPolygon(x2,y2,n2);
        int x3[]={148+v,168+v,178+v,158+v,148+v};
        int y3[]={259,259,290,295,259};
        int n3=5;j.fillPolygon(x3,y3,n3);
        j.drawOval(80+v,290,90,40);

        j.setColor(Color.YELLOW);  // color of legs circles

        j.fillOval(80+v,290,90,40);
        j.fillOval(140+v,290,80,40);
        j.setColor(Color.BLACK);
        j.drawOval(140+v,290,80,40);

        j.drawLine(170+v,240,180+v,230);
        j.drawLine(170+v,250,185+v,235);
        j.drawLine(180+v,250,190+v,240);

        j.drawArc(150+v,258,40,40,0,90);

}
```

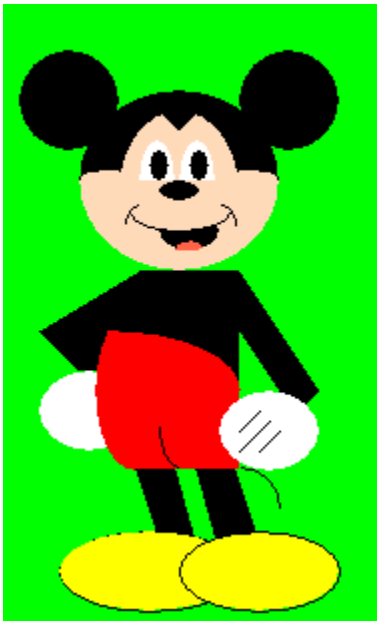

Figure 8-1 Mickey Mouse

9 Clock

```
public void paint(Graphics g) {
		// TODO Auto-generated method stub
		super.paint(g);

		//border clock
		if(status == 0){
			g.setColor(new Color(230, 136, 50));
			g.fillOval(25, spacing, 350, 350);
			g.setColor(Color.WHITE);
			g.fillOval(35, spacing+10, 330, 330);
		}else if(status == 1){
			g.setColor(Color.GRAY);
			g.fillOval(25, spacing, 350, 350);
			g.setColor(Color.WHITE);
			g.fillOval(35, spacing+10, 330, 330);
		}

		size = 400 -spacing;
		centerX = 400/2;
		centerY = 400/2+10;

		//clock face
		drawClockFace(g);

		//number clock face
		drawNumberClock(g);

		//get system time
		cal = Calendar.getInstance();
		hour = cal.get(Calendar.HOUR);
		minute = cal.get(Calendar.MINUTE);
		second = cal.get(Calendar.SECOND);

		//draw digital clock
		if(status==1){
			g.setColor(Color.LIGHT_GRAY);
			g.fillRect(centerX-40, centerY-35, 90, 20);
			g.setColor(Color.BLACK);
			g.drawRect(centerX-40, centerY-35, 90, 20);
			sf = new SimpleDateFormat("hh:mm:ss a");
			g.setColor(Color.BLACK);
			g.setFont(new Font("Tahoma", Font.BOLD, 12));
			g.drawString(sf.format(cal.getTime()), centerX-35+3, centerY-
35+15);
		}

		//draw date in clock 3
		if(status ==2){
			g.setColor(Color.ORANGE);
			g.drawRect(centerX+60, centerY-10, 40, 20);
			sf = new SimpleDateFormat("dd");
			g.drawString(sf.format(cal.getTime()), centerX+60+12,
centerY+5);
```

```
            }

            //draw hands
            if(status==2){

drawHands(g,hour,minute,second,colorSecond.RED,colorMHour.YELLOW);
            }else{

drawHands(g,hour,minute,second,colorSecond.RED,colorMHour.BLACK);
            }

            //draw point clock
            g.setColor(Color.BLACK);
            g.fillOval(centerX-5, centerY-5, 10, 10);
            g.setColor(Color.RED);
            g.fillOval(centerX-3, centerY-3, 6, 6);

    }
```

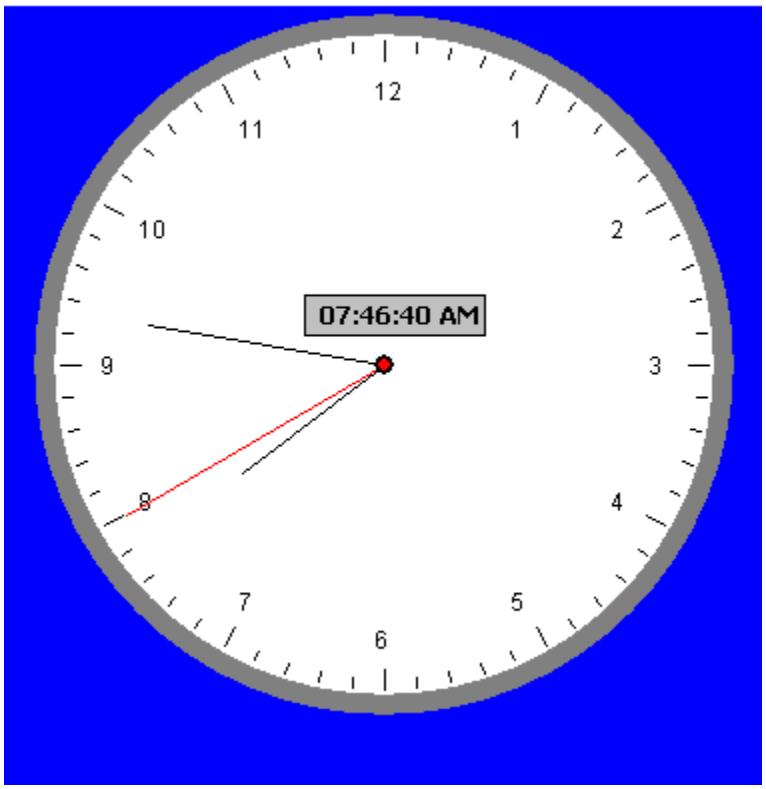

Figure 9-1 Clock