Digital Assignment – 2

ITE1005 - Advanced JAVA Programming

Slot – G2 + TG2
Faculty- Prof. Priya V

Submitted By:

Registration No.- 16BIT0453
Name- Krishna Kumar Mahto

Snake Game:

Applet features used:

1. Applet window

2. Packages used:

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.Image;
```

1. App description:

The application implements the classic Snake Game. The user can play the game with keyboard arrow keys- make the snake move left, right, up and down. The objective of the game is to feed the snake. The game screen will show the food as a square. With each successful feeding, the size of the snake increments by one unit. The game ends if the snake hits itself, or it hits the boundary of the screen. The user is shown a "Game Over" message with the score which is equal to the number of times the snake was able to eat food.

2. Program

<u>Program Description</u>

The program uses three custom abstract data types and one driver program:
i. Point
ii. Snake
iii. Token

Point data-type has been used to represent each unit of the snake's body. Many points placed adjacent to each other form a complete snake. Whenever the snake eats a food item, one point is added to its body so that its size increases.
The member variables:
private int x, y;

This abstract data type has following methods:
i. setX(int x)
ii. setY(int y)
iii. getX()

iv. getY()

The methods are self-explanatory.

Snake data type represents the snake. It has been implemented as a list of *Point* data type.

The member variables:

List<Point> snakePoints;
int xDir, yDir;
boolean isMoving, elongate;

It has the following methods:

i. public void draw(Graphics g)
ii. public void move()
iii. public boolean snakeCollision()
iv. public boolean isMoving()
v. public void setIsMoving(boolean b)
vi. public int getXDir()
vii. public int getYDir()
viii. public void setXDir(int x)
ix. public void setYDir(int y)
x. public int getHeadX()
xi. public int getHeadY()
xii. public void setElongate(boolean b)

Most of the methods are self-explanatory. getXDir() gives the direction in which the snake moves if it moves along horizontal axis (x-axis). Similar is the function of getYDir().
The member variable xDir, if it is equal to 1 then it has been defined to move in the x-drection, if it is set to -1, the snake is defined to be moving left.
Similarly, yDir = 1 means the snake is moving downwards, and if yDir = -1, then the snake is moving upwards.
getHeadX() method returns the x-position of the head of the snake, and getHeadY() returns the y-position of the head of the snake.

Token data type represents food for the snake.

Data members:

i. private int x, y, score;
ii. private Snake snake;

Methods:

i. public void changePosition()
ii. public int getScore()
iii. public void draw(Graphics g)
iv. public boolean snakeCollision()

snakeCollision() methods is used to check if the snake collided with the token (food). If it did, it increments the score, and elongates the snake.

The app is driven by an applet which is also the driver program. It makes use of all the data types, ie, Point, Snake and Token. These data types have been defined as Java classes. The name of the applet is SnakeGame. It implements the Runnable interface and the KeyListener interface. In order to inherit Applet properties, it extends the Applet class.

Methods:

i. public void init()
ii. public void paint(Graphics g)
iii. public void update(Graphics g)
iv. public void repaint(Graphics g)
v. public void run()
vi. public void checkGameOver()
vii. public void keyPressed(KeyEvent kEvent)

init(), paint(), update(), repaint() are common applet methods. CheckGameOver() method check if the game is over and sets gameOver variable to true if the conditions of gave over are satisfied.
The second method is abstract method inherited from the KeyListener interface. Based on which key is pressed, it assigns a direction to the snake by modifying the xDir and yDir variables.

Codes:

SnakeGame.java

```
// <applet code = "SnakeGame" width = "400" height =
"400"></applet>

import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.Image;

public class SnakeGame extends Applet implements Runnable,
KeyListener {

  // Setting up reference variables to support double buffer for
graphics..\
  // rendering and updation.
  Graphics graphics;
  Image img;
  Thread thread;
  Snake snake;
  boolean gameOver;
  Token token;

  public void init() {
    this.resize(400, 400);
    boolean gameOver = false;
    img = createImage(400, 400);
    graphics = img.getGraphics(); // this graphics is currently
not a part of Applet window
                                  // the graphics object is the
graphics context for drawing an off-screen image

    this.addKeyListener(this);
    snake = new Snake();
    token = new Token(snake);
    thread = new Thread(this);
    thread.start();
  }

  public void paint(Graphics g) {
    /* setColor() sets this graphics context's current color to
the specified..\
     color. All subsequent graphics operations using this graphics
context use..\
     this specified color. */
    graphics.setColor(Color.black);
    graphics.fillRect(0,0,400,400);
    if(!gameOver) {
      snake.draw(graphics);
      token.draw(graphics);
    }
    else {
      graphics.setColor(Color.red);
      graphics.drawString("Game Over", 180, 150);
      graphics.drawString("Score: " + token.getScore(), 180, 170);
    }

    g.drawImage(img, 0, 0, null); // g is the reference variable
containing the context\
                                  // of the applet window.
  }

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

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

  public void run() {
    for(;;) {

      if(!gameOver) {
        snake.move();
        this.checkGameOver();
        token.snakeCollision();
      }
      this.repaint();

      try {
        Thread.sleep(40); // to allow our eyes to pick up that the
snake moved
      } catch (InterruptedException ex) {
        ex.printStackTrace();
      }
    }
  }

  public void checkGameOver() {
    if(snake.getHeadX() < 0 || snake.getHeadX() > 396)
      gameOver = true;
    if(snake.getHeadY() < 0 || snake.getHeadY() > 396)
      gameOver = true;
    if(snake.snakeCollision())
      gameOver = true;
  }
  public void keyPressed(KeyEvent kEvent) {

    if(!snake.isMoving()) {
      if(kEvent.getKeyCode() == KeyEvent.VK_UP ||
kEvent.getKeyCode() == KeyEvent.VK_RIGHT || kEvent.getKeyCode() ==
KeyEvent.VK_DOWN) {
        snake.setIsMoving(true);
      }
    }

    if(kEvent.getKeyCode() == KeyEvent.VK_UP) {
      if(snake.getYDir() != 1) { // if the snake is not going
down, then only it can go up
        snake.setYDir(-1);
        snake.setXDir(0); // while going up, snake can only go up.
Not (up + left, ie, diagonally) or anything like that
      }
    }
    if(kEvent.getKeyCode() == KeyEvent.VK_DOWN) {
      if(snake.getYDir() != -1) {
        snake.setYDir(1);
        snake.setXDir(0);
      }
    }
    if(kEvent.getKeyCode() == KeyEvent.VK_LEFT) {
      if(snake.getXDir() != 1) {
        snake.setXDir(-1);
        snake.setYDir(0);
      }
    }
    if(kEvent.getKeyCode() == KeyEvent.VK_RIGHT) {
      if(snake.getXDir() != -1) {
        snake.setXDir(1);
        snake.setYDir(0);
      }
    }
  }

  public void keyReleased(KeyEvent kEvent) {

  }
  public void keyTyped(KeyEvent kEvent) {

  }
}
```

Point.java:

```
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 this.x;
  }

  public int getY() {
    return this.y;
  }
}
```

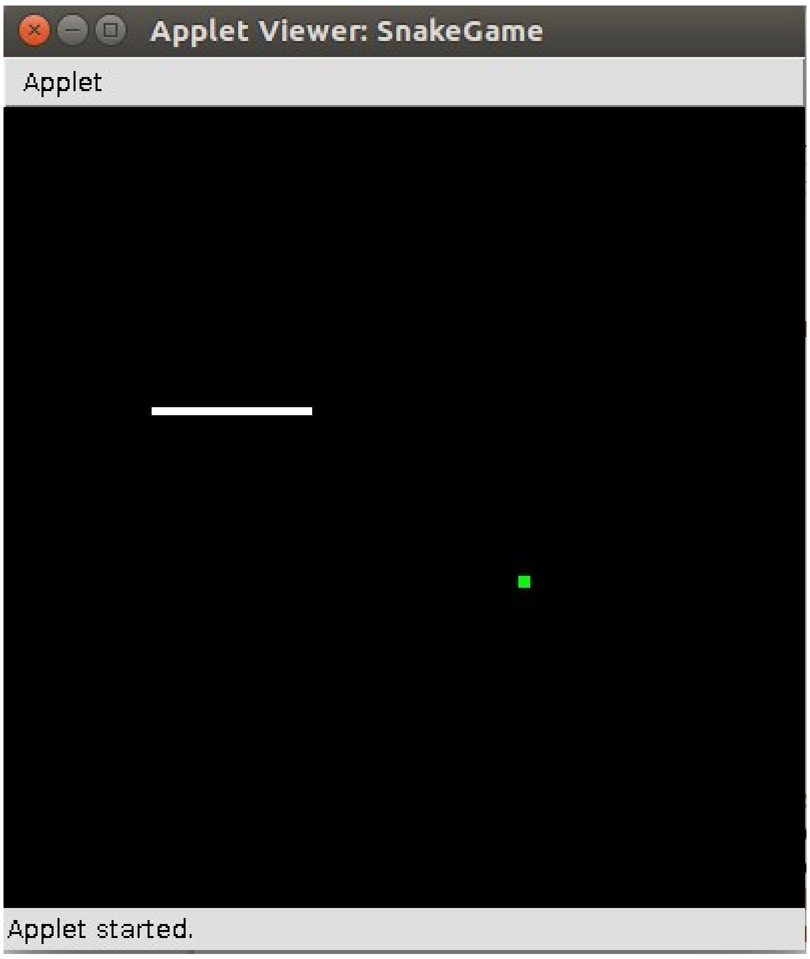
Snake.java

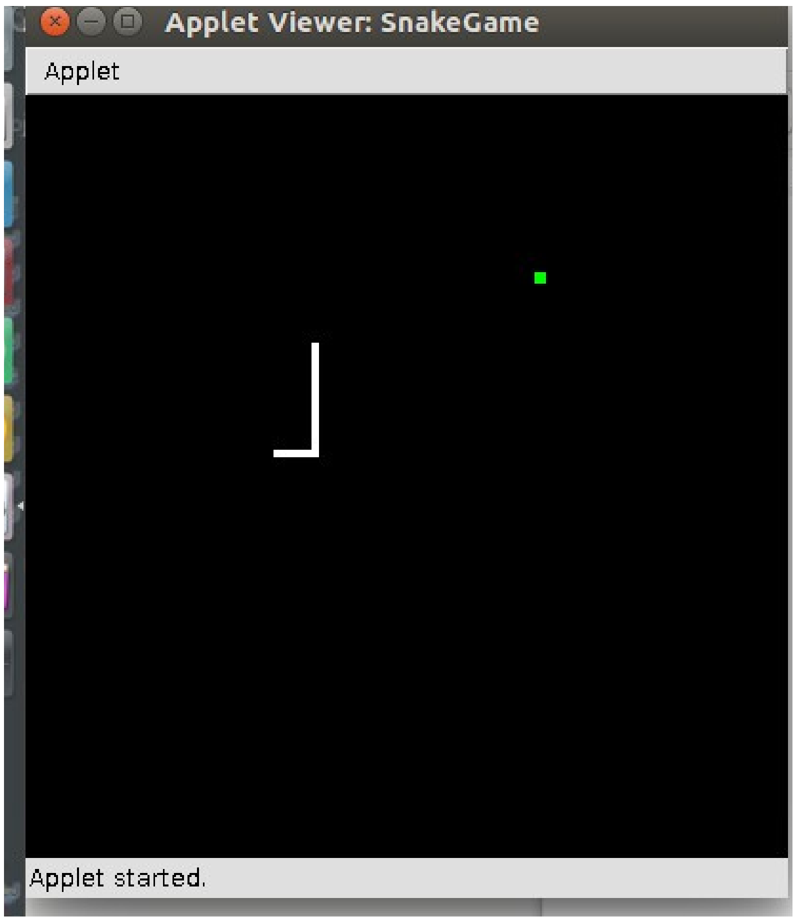
```
// Our Snake is going to be a list of points.
import java.util.ArrayList;
import java.util.List;
import java.awt.Graphics;
import java.awt.Color;

public class Snake {
  List<Point> snakePoints;
  int xDir, yDir; // if xDir == -1,snake heads towards the left,
if xDir == 1, snake moves towards the right
                  // if yDir == -1, snake moving up.........." "
"
                  // if xDir == 0, snake not moving in xDir, if
yDir == 0, snake not moving in y direction.
  boolean isMoving, elongate;
  final int STARTSIZE = 20, STARTX = 150, STARTY = 150;

  public Snake() {
    snakePoints = new ArrayList<Point>();

    // Snake not moving initially
    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));
  }

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

  public void move() {
    if(isMoving) {
      Point head = snakePoints.get(0);
      Point last = snakePoints.get(snakePoints.size() - 1);
      Point newHead = new Point(head.getX() + xDir * 4,
head.getY() + yDir * 4);
      for(int i = snakePoints.size()-1; i >= 1; i--) {
        snakePoints.set(i, snakePoints.get(i - 1));
      }
      snakePoints.set(0, newHead);
      if(elongate) {
        snakePoints.add(last);
        elongate = false;
      }
    }
  }

  public boolean snakeCollision() {
    int x = this.getHeadX();
    int y = this.getHeadY();

    for(int i = 1; i < snakePoints.size(); i++) {
      if(snakePoints.get(i).getX() == x &&
snakePoints.get(i).getY() == y)
        return true;
    }
    return false;
  }

  public boolean isMoving() {
    return isMoving;
  }

  public void setIsMoving(boolean b) {
    isMoving = b;
  }

  public int getXDir() {
    return this.xDir;
  }

  public int getYDir() {
    return this.yDir;
  }

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

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

  // X position of the head of snake
  public int getHeadX() {
    return this.snakePoints.get(0).getX(); // invokes getX()
method defined in Point class
                                          // since 0th element is
a Point object
  }

  public int getHeadY() {
    return this.snakePoints.get(0).getY();
  }
  public void setElongate(boolean b) {
    elongate = b;
  }
}
```

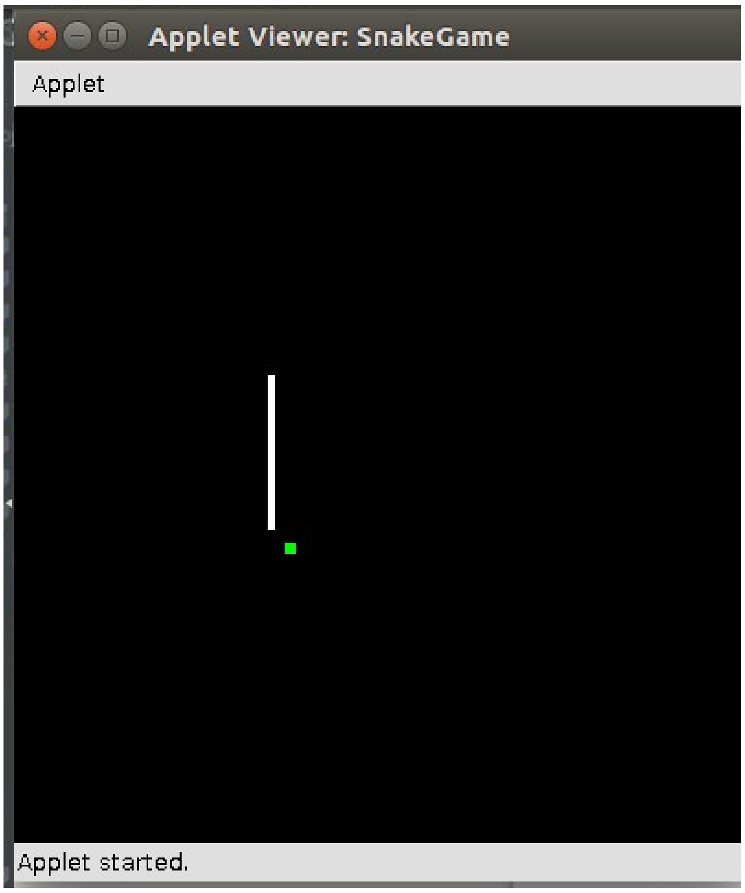
Token.java:

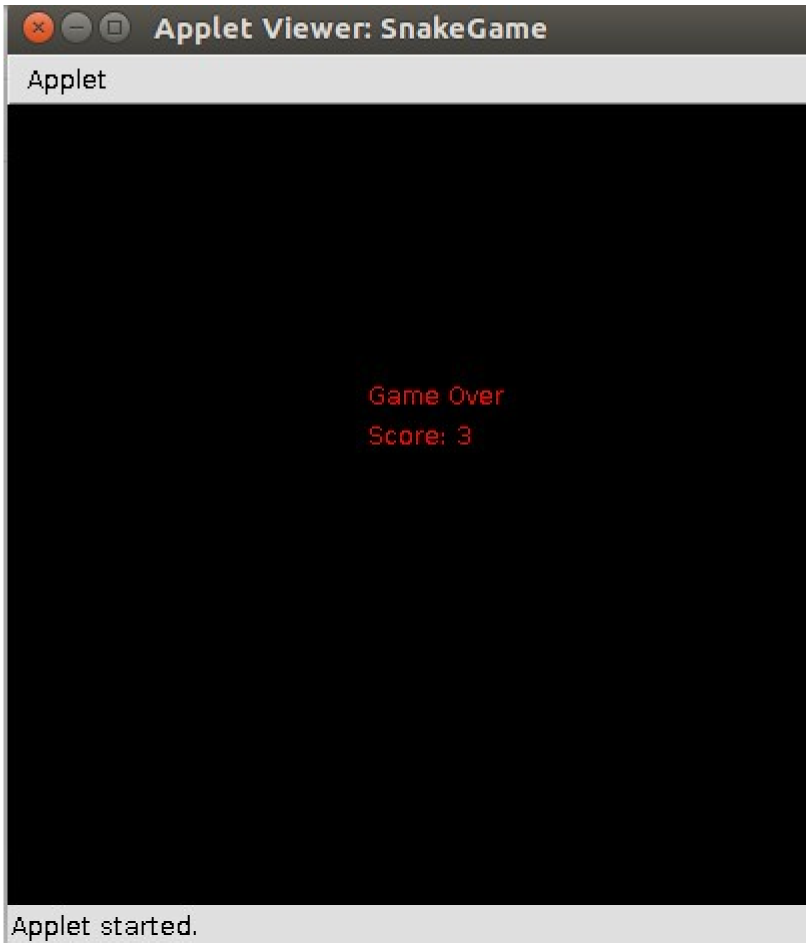
```
import java.awt.Graphics;
import java.awt.Color;

public class Token {
  private int x, y, score;

  private Snake snake;

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

  public void changePosition() {
    x = (int)(Math.random() * 395);
    y = (int)(Math.random() * 395);
  }

  public int getScore() {
    return score;
  }

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

  public boolean snakeCollision() {
    int snakeX = snake.getHeadX() + 2;
    int snakeY = snake.getHeadY() + 2;
    if(snakeX >= x-1 && snakeX <= (x+7))
      if(snakeY >= y-1 && snakeY <= (y+7)) {
        changePosition();
        score++;
        snake.setElongate(true);
        return true;
      }

    return false;
  }
}
```

Output Screenshots:









```
Game Over
Score: 3
```