Unit 3: Graphics Class 3

Unit Structure

3.1 LEARNING OBJECTIVE

The objective of this unit is to make the students,

- To learn, understand and define graphics class and its methods
- To learn, understand and define the Font class and its methods
- To learn, understand and define the Color class and its methods
- To learn, understand the arrangement of AWT controls on the container
- To learn, understand different Layout and its parameters

3.2 OUTCOMES

After learning the contents of this chapter, the students will be able to:

- Use Graphics class and its various methods
- Use Font class and its various methods in programs
- Use Color class and its various methods in programs
- Write a graphical application using graphics, font and color class
- Use Layout Manager to arrange AWT components on the containers

3.3 INTRODUCTION

Java provides the platform to develop graphics based application using the Graphics class. This unit dicusses various java functionalities for painting shapes like rectangle, polygon etc. The unit covers the use of color and fonts. It also demonstrates the filling of object once it is drawn on the container. It also discusses various font family, its style to display the content on the container. It is essential to learn to beautify the components placed on the container area using Font and Color class. Withour the proper arrangement of control on the containers the GUI of the application looks jagged. So, it becomes very important for the programmer to arrangement the controls on the containers. Here, the Layout Manager comes. This unit also discusses different layout techniques to arrange components on the containers. It also covers various techniques to arrange the control manually using setBounds method.

3.4 GRAPHICS CLASS

In the AWT package, the Graphics class provides the foundation for all graphics operations. At one end the graphics context provides the information about drawing operations like the background and foreground colors, font and the location and dimensions of the region of a component. At the other end, the Graphics class provides methods for drawing simple shapes, text, and images at the destination.

To draw any object a program requires a valid graphics context in the form of instance of the Graphics class. Graphics class is an abstract base class, it cannot be instantiated. An instance is created by a component and handed over to the program as an argument to a component's update() and paint() methods. The update() and paint() method should be redefined to perform the desired graphics operations. There are various methods used for drawing different component on the containser. Thay are discussed below.

- **repaint() Method**

The repaint() method requests for a component to be repainted. This method has various forms as shown below:

1. public void repaint();
2. public void repaint(long tm) ; // Specify a period of time in milliseconds

Once a period of time is provided, the painting operation will occur before the time elapses.

3. public void repaint(int x, int y, int w, int h);

We can also provide that only a portion of a component be repainted. It is useful when the paint operation is time-consuming, and only a portion of the display needs to be repainted.

4. public void repaint(long tm, int x, int y, int w, int h);

- **public void update(Graphics g)**

The update() method is called in turn to a repaint() request. This method takes an instance of the Graphics class as an argument. The scope of graphics instance is valid only within the context of the update() method and the methods it calls. The default implementation of the Component class will erase the background and calls the paint() method.

- **public void paint(Graphics g)**

The paint() method is called from an update() method, and is responsible for drawing the graphics. It takes an instance of the Graphics class as an argument.

- **void drawLine(int xStart, int yStart, int xStop, int yStop)**

It draws a straight line, a single pixel wide, between the specified start and end points. The line will be drawn in the current foreground color. This methods works when invoked on a valid Graphics instance and used only within the scope of a component's update() and paint() methods.

- **Retangle**

Rectangle object can be drawn in different ways like,

1. void drawRect(int x, int y, int width, int height)
2. void fillRect(int x, int y, int width, int height)
3. void drawRoundRect(int x, int y, int width, int height, int arcwidth, int archeight)
4. void fillRoundRect(int x, int y, int width, int height, int arcwidth, int archeight)
5. void draw3DRect(int x, int y, int width, int height, boolean raised)
6. void fill3DRect(int x, int y, int width, int height, boolean raised)

All the method requires, the x and y coordinates as parameters to start the rectangle, and the width and height of the rectangle. The width and height must be positive values. Rectangles can be drawn in three different styles: plain, with rounded corners, and with a three-dimensional effect (rarely seen).

The RoundRect methods require an arc width and arc height to control the rounding of the corners. The 3 dimensional methods require an additional parameter that indicates whether or not the rectangle should be raised. These all method works when invoked on a valid Graphics instance and used only within the scope of a component's update() and paint() methods.

- **Ovals and Arcs**

Ovals and Arc object can be drawn in different ways like,

1. void drawOval(int x, int y, int width, int height)
2. void fillOval(int x, int y, int width, int height)
3. void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
4. void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

Each of this method requires, the x and y coordinates of the center of the oval or arc, and the width and height of the oval or arc. The width and height must be positive values. The arc methods require a start angle and an arc angle, to specify the beginning of the arc and the size of the arc in degrees.

This methods works when invoked on a valid Graphics instance and used only within the scope of a component's update() and paint() methods.

- **Polygons**

Polygon object can be drawn in different ways like,

1. void drawPolygon(int xPoints[], int yPoints[], int nPoints)
2. void drawPolygon(Polygon p)
3. void fillPolygon(int xPoints[], int yPoints[], int nPoints)
4. void fillPolygon(Polygon p)

Polygons object drawn from a sequence of line segments. Each of this method requires, the coordinates of the endpoints of the line segments that will make the polygon. These endpoints can be specified by first, the two parallel arrays of integers, one representing the x coordinates and the other representing the y coordinates; second is, using an instance of the Polygon class. The Polygon class provides the method addPoint(), which allows a polygon to be organized point by point. These methods works when invoked on a valid Graphics instance and used only within the scope of a component's update() and paint() methods.

3.4.1 COLOR CLASS

The java.awt.Color class provides 13 standard colors as constants. They are: RED, GREEN, BLUE, MAGENTA, CYAN, YELLOW, BLACK, WHITE, GRAY, DARK_GRAY, LIGHT_GRAY, ORANGE and PINK. Colors are created from red, green and blue components of RGB values. The range of RGB will be from 0 to 255 or floating point values from 0.0 to 1.0. We can use the toString() method to print the RGB values of these color (e.g., System.out.println(Color.RED)):

- **Methods**

To implement color in objects or text, two Color methods getColor() and setColor() are used. Method getColor() returns a Color object and setColor() method used to sets the current drawing color.

Now check below program to learn how these methods can be used.

Example:

```
import java.awt.Frame;
import java.awt.Panel;
import java.awt.Graphics;
import java.awt.Polygon;
import java.awt.Color;
public class PictureDraw extends Panel
{
        public void paint(Graphics g)
        {
                //Print a String message
                g.drawString("Welcome to BAOU", 20, 20);
                //draw a Line
                g.drawLine(0, 0, 100, 70);
                //draw a Oval
                g.drawOval(100, 100, 100, 100);
                //draw a rectangle
                g.drawRect(80, 80, 125, 125);
                //draw a Polygon
                int x[] = {35, 155, 35, 155, 35};
                 int y[] = {35, 35, 155, 155, 35};
                 g.drawPolygon(x,y,5);   //points = 5;
                g.setColor(Color.orange);
                Polygon pg = new Polygon();
                  pg.addPoint(220, 30);
                  pg.addPoint(300, 35);
                  pg.addPoint(320, 95);
                  pg.addPoint(275, 70);
                  pg.addPoint(210, 100);
                  pg.addPoint(180, 50);
                  g.drawPolygon(pg);
                g.fillPolygon(pg);
        }
        public static void main(String[] args)
        {
                Frame f= new Frame("Graphics Control");
                f.add(new PictureDraw());
                f.setSize(600, 1500);
                f.setVisible(true);
                f.setResizable(false);
        }
}
```

Output:

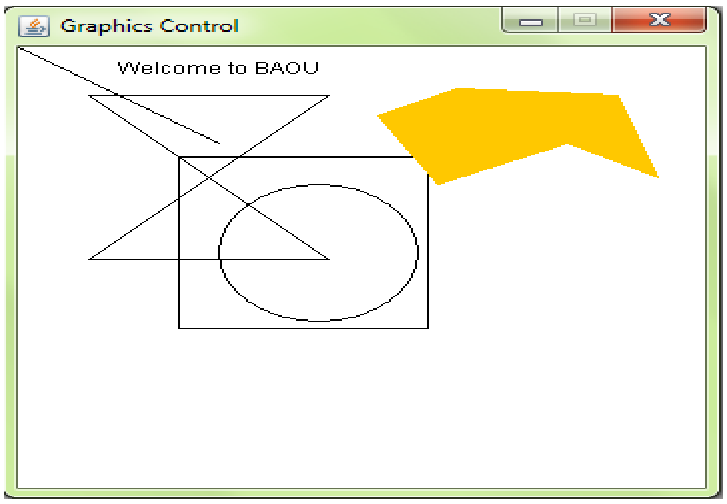


Figure-115: Output of program

➢ **Check Your Progress 1**

1) Write all state information that Graphics object encapsulates.

……………………………………………………………………………………

……………………………………………………………………………………

2) Write two important roles of Graphics class.

……………………………………………………………………………………

……………………………………………………………………………………

3) How does a Color class create color?

……………………………………………………………………………………

……………………………………………………………………………………

4) State the relationship between the Canvas and Graphics class.

……………………………………………………………………………………

……………………………………………………………………………………

3.4.2 FONT CLASS

The java.awt.Font class represents a method of specifying and using fonts. That font will be used to render the texts. The Font class constructor is used to construct a font object using the font's name, style (PLAIN, BOLD, ITALIC, or BOLD + ITALIC) and font size. In java, fonts are named in a platform independent fashion and then mapped to local fonts that are supported by the underlying operating system. The getName() method is used to return the logical Java font name of a particular font and the getFamily() method is used to return the operating system-specific name of the font. In java the standard font names are Courier, Helvetica, TimesRoman etc. There are 3 logical font names. Java will select a font name in the system that matches the general feature of the logical font.

I. Serif: This is often used for blocks of text (example, Times).

II. Sansserif: This is often used for titles (example, Arial or Helvetica).

III. Monospaced: This is often used for computer text (example, Courier).

The logical font family names are “Dialog”, “DialogInput”, “Monospaced”, “Serif”, or “SansSerif” and Physical font names are actual font libraries such as

“Arial”, “Times New Roman” in the system. There is logical font names, standard on all platforms and are mapped to actual fonts on a particular platform.

- **Constructor:**

public Font(String fontName, int fontStyle, int fontSize);

where, fontName represents Font Family name

fontStyle represents Font.PLAIN, Font.BOLD, Font.ITALIC or Font.BOLD or Font.ITALIC

fontSize represents the point size of the font (in pt) (1 inch has 72 pt).

The setFont() method to set the current font for the Graphics context g for rendering texts.

For example,

```
g.drawString(“Welcome to BAOU”, 15, 25);     // in default font

Font fontTest = new Font(Font.SANS_SERIF, Font.ITALIC, 15);

g.setFont(fontTest);

g.drawString(“Gujarat Vidyapith”, 10, 50);  // in fontTest
```

We can use GraphicsEnvironment’s getAvailableFontFamilyNames() method to list all the font family names; and getAllFonts() method to construct all Font instances (font size of 1 pt).

For example,

```
GraphicsEnvironment fontEnv =
GraphicsEnvironment.getLocalGraphicsEnvironment();
String[] fontList = fontEnv.getAvailableFontFamilyNames();
for (int i = 0; i < fontList.length; i++)
{
    System.out.println(fontList [i]);
}
// Construct all Font instance (with font size of 1)
```

```
Font[] fontList = fontEnv.getAllFonts();
for (int i = 0; i < fontList.length; i++)
{
     System.out.print(fontList [i].getFontName() + " : ");
     System.out.print(fontList [i].getFamily() + " : ");
     System.out.print(fontList [i].getName());
}
```

Example:

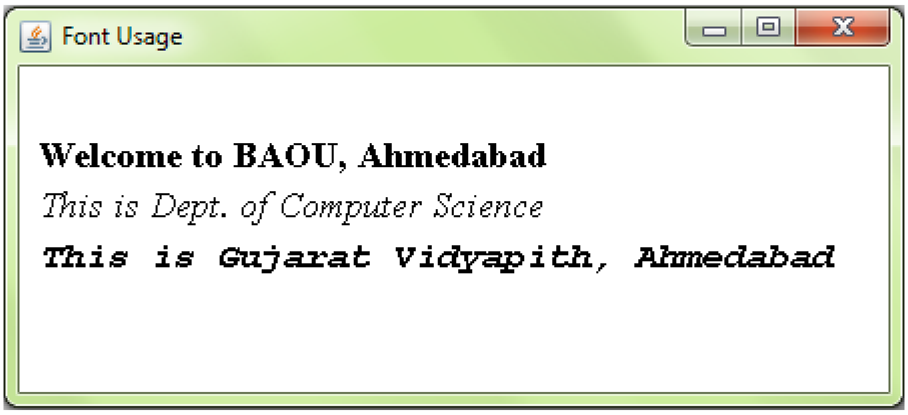
Now check below program to learn how Font class and its method can be used.

```
import java.awt.Font;
import java.awt.Frame;
import java.awt.Panel;
import java.awt.Graphics;
public class FontClass extends Panel
{
   public void paint(Graphics g)
        {
      Font f = new Font("Arial", Font.PLAIN, 18);
      Font fb = new Font("TimesRoman", Font.BOLD, 18);
      Font fi = new Font("Serif", Font.ITALIC, 18);
      Font fbi = new Font("Monospaced", Font.BOLD + Font.ITALIC, 18);

      g.setFont(f);
      g.setFont(fb);
      g.drawString("Welcome to BAOU, Ahmedabad", 10, 50);
      g.setFont(fi);
      g.drawString("This is Dept. of Computer Science", 10, 75);
      g.setFont(fbi);
      g.drawString("This is Gujarat Vidyapith, Ahmedabad", 10, 100);
   }
        public static void main(String s[])
        {
                Frame f= new Frame("Font Usage");
```

```
                    f.add(new FontClass());
                    f.setVisible(true);
                    f.setSize(1550,200);
          }
}
```

Output:



```
Welcome to BAOU, Ahmedabad
This is Dept. of Computer Science
This is Gujarat Vidyapith, Ahmedabad
```

Figure-116: Output of program

➢ **Check Your Progress 2**

1) How many ways can user display Font style?

………………………………………………………………………………………

………………………………………………………………………………………

2) What is the difference between paint() and repaint() methods?

………………………………………………………………………………………

………………………………………………………………………………………

3) Discuss different Font class methods.

………………………………………………………………………………………

………………………………………………………………………………………

4) Differentiate the Font and FontMetrics classes.

………………………………………………………………………………………

………………………………………………………………………………………

3.5 LAYOUT MANAGER

It is possible to position and size the GUI component by hard coding but also challenging and therefore not advised. So, it is advised to use layout manager as it is easier to adjust and rework positions, sizes and the overall look-and-feel of the container. Use of layout managers facilitates a top-level or base container to have its own layout while other containers on top of it have their own layout which is completely independent. Whenever we add any components to a container, the final configuration of size and positioning is ultimately decided by the layout manager of the underlying container. Therefore, anytime a container is resized, its layout manager has to position each of the components within it. JPanel and content panes are the containers base of the GUI application structure and belong to FlowLayout and BorderLayout classes. It is recommended to set layout manager of the container.

LayoutManager is an interface. It is implemented by all the classes of layout managers. The following class represents the layout managers from java.awt package.

1. BorderLayout
2. FlowLayout
3. GridLayout
4. CardLayout
5. GridBagLayout

3.5.1 BORDERLAYOUT

The BorderLayout helps to arrange the components in north, south, east, west and center regions. This is the default layout for frame or window. The BorderLayout has five constants for each region. They are public static final int NORTH, SOUTH, EAST, WEST, CENTER.

Constructors:

1. BorderLayout(): This allows us to create a border layout without gaps between the components.
2. JBorderLayout(int hgap, int vgap): This allows us to create a border layout with the given horizontal and vertical gaps between the components.

Note: In this unit, we have used Frame as the main container in all programs.

Example: The following program depicts the use of BorderLayout.

```
import java.awt.*;
public class BorderLout extends Frame
{
        BorderLout(String title)
        {
                super(title);
                Button b1=new Button("BAOU");;
                Button b2=new Button("GVP");;
                Button b3=new Button("DCS");;
                Button b15=new Button("BCA");;
                Button b5=new Button("MCA");;

                add(b1,BorderLayout.NORTH);
                add(b2,BorderLayout.SOUTH);
                add(b3,BorderLayout.EAST);
                add(b15,BorderLayout.WEST);
                add(b5,BorderLayout.CENTER);
        }
        public static void main(String[] args)
        {
                Frame bly=new BorderLout("Border");
                bly.setSize(300,300);
                bly.setVisible(true);
        }
}
```

Output:

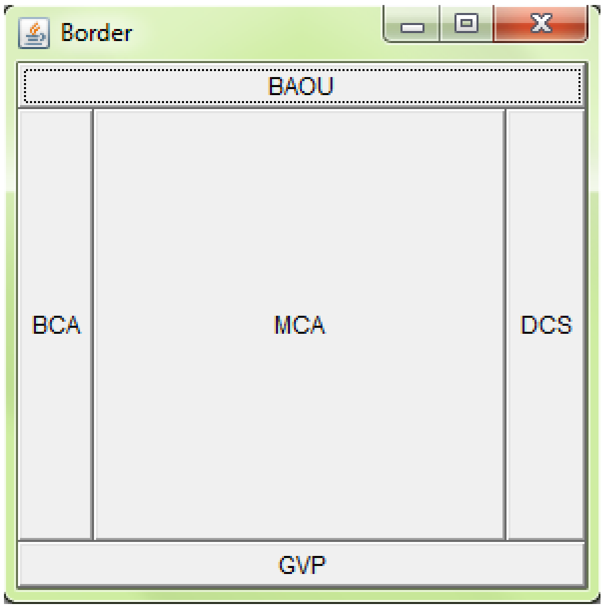


Figure-117: Output of program

3.5.2 FLOWLAYOUT

The FlowLayout is used to arrange the components in a line. As we keeps adding components, it arranges them one after another from left to right in a flow. This layout is the default layout of applet or panel.

Constants of FlowLayout:

There are total five constants used in FlowLayout. They are public static final int LEFT, RIGHT, CENTER, LEADING and TRAILING.

Constructors:

1. **FlowLayout():** It allows us to create a flowlayout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** It allows us to create creates a flowlayout with the specified alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** It allows us to create a flowlayout with the specified alignment and horizontal and vertical gap.

Example: The following program depicts the use of FlowLayout.

```
import java.awt.*;
public class FlowLout extends Frame
{
        FlowLout(String title)
        {
                super(title);
                Button b1=new Button("BAOU");
                Button b2=new Button("GVP");
                Button b3=new Button("DCA");
                Button b15=new Button("MCA");
                Button b5=new Button("BCA");

                add(b1);add(b2);add(b3);add(b15);add(b5);
                //setting flow layout of right alignment
                setLayout(new FlowLayout(FlowLayout.RIGHT));

}
public static void main(String[] args) {
   Frame fly=new FlowLout("Flow");
   fly.setSize(250,200);
   fly.setVisible(true);
}
}
```

Output:

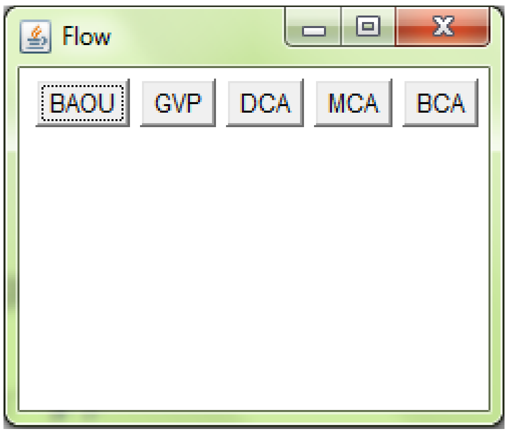


Figure-118: Output of program

3.5.3 GRIDLAYOUT

The GridLayout helps us to arrange the components in rectangular grid. Only one component will be displayed in each rectangle.

Constructors:

1. GridLayout(): This constructor allows us to create a gridlayout with one column per component in a row.
2. GridLayout(int rows, int columns): This constructor allows us to create a gridlayout with the specified rows and columns but without the gaps between the components.
3. GridLayout(int rows, int columns, int hgap, int vgap): This constructor allows us to create a gridlayout with the specified rows, columns, horizontal gap and vertical gap.

Example: The following program depicts the use of GridLayout.

```
import java.awt.*;
public class GridLout extends Frame
{
GridLout(String title){
   super(title);
   Button ba=new Button("A");
   Button bb=new Button("B");
   Button bc=new Button("C");
   Button bd=new Button("D");
   Button be=new Button("E");
   Button bf=new Button("F");
   Button bg=new Button("G");
   Button bh=new Button("H");
   Button bi=new Button("I");

   add(ba);add(bb);add(bc);add(bd);add(be);
   add(bf);add(bg);add(bh);add(bi);
 //setting gridlayout of 3 rows and 3 columns
```

```
    setLayout(new GridLayout(3,3));
}
public static void main(String[] args) {
    Frame fyl=new GridLout("Grid");
    fyl.setSize(300,300);
    fyl.setVisible(true);
}
}
```

Output:

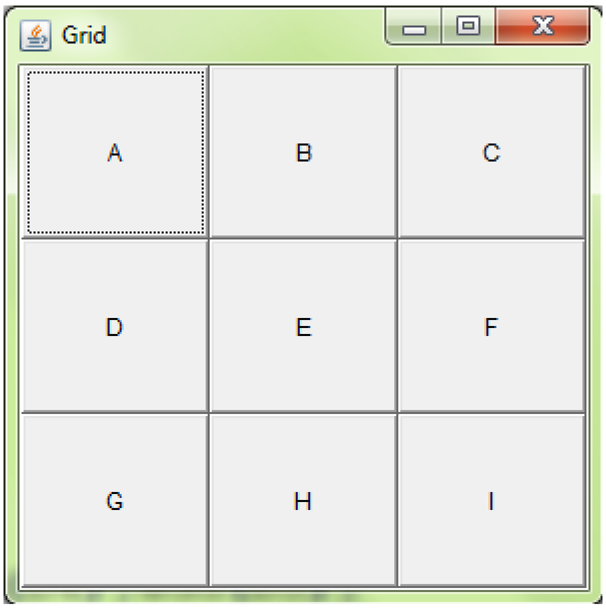


Figure-119: Output of program

3.5.4 CARDLAYOUT

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout. There are various methods like next, first, previous, last and show to flip from one card to another card.

Constructors:

1. CardLayout(): This constructor allows us to create a cardlayout with zero horizontal and vertical gap.

2. CardLayout(int hgap, int vgap): This constructor allows us to create a cardlayout with the specified horizontal and vertical gap.

Example: The following program depicts the use of GridLayout. We have used three panels as a card to show different pane.

```
Import java.awt.*;
import java.awt.event.*;
class CardLout extends Frame implements ActionListener {
    CardLayout cardlt = new CardLayout(25,25);
    CardLout(String str)  {
        super(str);
        setLayout(cardlt);
        Button Panel1 = new Button("BAOU");
        Button Panel2 = new Button ("DCS");
        Button Panel3 = new Button("GVP");
        add(Panel1,"BAOU");
        add(Panel2,"DCS");
        add(Panel3,"GVP");
        Panel1.addActionListener(this);
        Panel2.addActionListener (this);
        Panel3.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)
    {
      cardlt.next(this);
    }

   public static void main(String args[])
   {
           CardLout frame = new CardLout("CardLayout");
           frame.setSize(210,170);
           frame.setResizable(false);
           frame.setVisible(true);
   }
}
```

Output:

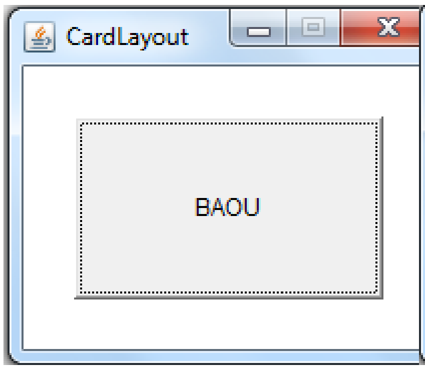


Figure-120: Output of program

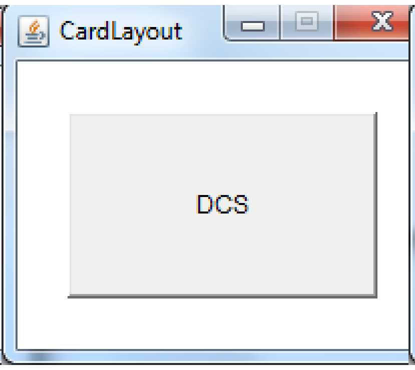


Figure-121: Output of program

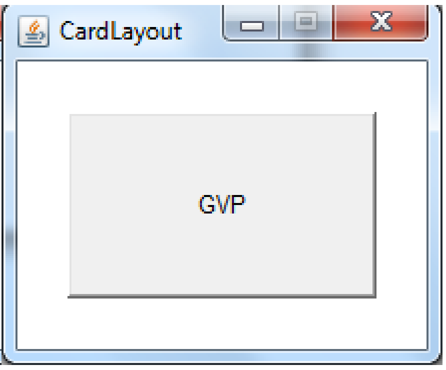


Figure-122: Output of program

3.5.5 GRIDBAGLAYOUT

The Java GridBagLayout class helps to align components vertically, horizontally or along their baseline. It is also most flexible as well as complex layout managers. It places components in a grid of rows and columns, allowing particular components to span multiple rows or columns. Not all rows and columns necessarily have the same height. It places components in cells in a grid and then uses the components' preferred sizes to determine how big the cells should be to contain component.

To use a GridBagLayout effectively, we need to customize one or more component’s GridBagConstraints. By setting one of its instance variables we can customize a GridBagConstraints object. The instances are:

- gridx, gridy

  This variables specifies the cell at the top most left of the component's display area, where address gridx=0 refers the leftmost column and address gridy=0 refers the top row. GridBagConstraints.RELATIVE is the default value. It specifies that the component placed just to the right of (gridx) or below (gridy) the component.

- gridwidth, gridheight

  This variable specifies the number of cells in a row (for gridwidth) or column (for gridheight) in the component's display area. The default value is 1.

- fill

  This variable is used when the component's display area is larger than the component's requested size to decide whether to resize the component. We can pass NONE (default), HORIZONTAL (will not change its height), VERTICAL (will not change its width) and BOTH (component fill its display area entirely) with GridBagConstrain as valid values of fill.

- ipadx, ipady

  This variable specifies the internal padding. The width of the component will be its minimum width plus ipadx*2 pixels (as the padding applies to both sides of the component). Similarly, the height of the component will be its minimum height plus ipady*2 pixels.

- insets

  This variable specifies the external padding of the component. It will be the minimum amount of space between the component and the edges of its display area.

- anchor

  This variable is helps us when the component is smaller than its display area to decide where to place the component. We can pass CENTER (the default), NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST and NORTHWEST as valid values.

- weightx, weighty

  This variable is used to determine how to distribute space when we want to specify resizing behaviour or change of dimension.

Example: Below example uses GridBagConstraints instance for all the components the GridBagLayout manages. In real-life, it is recommended that you do not reuse GridBagConstraints. In the example, just before each component is added to the container, the code sets the appropriate instance variables in the GridBagConstraints object. Then after it adds the component to its container, passing the GridBagConstraints object as the second argument to the add method.

```
import java.awt.*;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;

public class gridBagLout extends Frame
{
   Button first, second,third,forth,fifth,sixth;
        public static void main(String[] args)
        {
      Frame gbl = new gridBagLout(“GridBag Layout”);
                gbl.setSize(300, 300);
                gbl.setVisible(true);
        }
   public gridBagLout(String str)
        {
           super(str);
          first=new Button("BAOU");
        second=new Button("DCS");
        third=new Button("MCA");
        forth=new Button("GVP");
        fifth=new Button("Ahmedabad");
        sixth=new Button("Gujarat");

   GridBagConstraints gbc = new GridBagConstraints();
   GridBagLayout layout = new GridBagLayout();
   setLayout(layout);

   gbc.fill = GridBagConstraints.HORIZONTAL;
   gbc.gridx = 0;
   gbc.gridy = 0;
   add(first, gbc);
   gbc.gridx = 1;
   gbc.gridy = 0;
   add(second, gbc);
   gbc.fill = GridBagConstraints.HORIZONTAL;
```

```
    gbc.ipady = 30;
    gbc.gridx = 0;
    gbc.gridy = 1;
    add(third, gbc);
    gbc.gridx = 1;
    gbc.gridy = 1;
    add(forth, gbc);
    gbc.gridx = 0;
    gbc.gridy = 2;
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridwidth = 2;  //Merge two columns
    add(fifth, gbc);
    gbc.gridx = 0;
    gbc.gridy = 3;
    gbc.gridwidth = 2;  //Merge two columns
    add(sixth, gbc);
    }
}
```

Output:

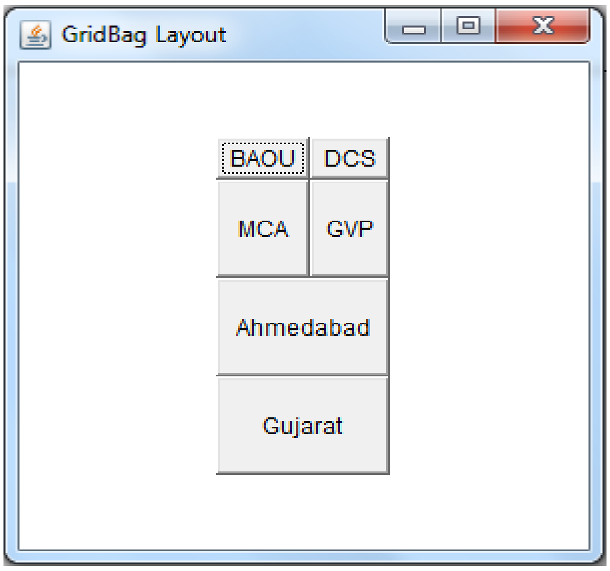


Figure-123: Output of program

➢ setBounds() method

setBounds() method of awt.component class is used to set the size and position of component. When we need to change the size and position of component then we can use this method

Syntax:

public void setBounds(int x, int y, int width, int height)

This parameter puts the upper left corner at location (x, y), where x is the number of pixels from the left of the screen and y is the number from the top of the screen.

Example:

```
import java.awt.*;
public class Setbound extends Frame
{
   Label name;
   TextField user;
   Button login;
   Setbound(String str)
        {
          super(str);
          setLayout(null);
          name=new Label("User_Name:");
          user=new TextField(10);
          login=new Button("Login");

          name.setBounds(50, 50, 75, 30 );
          add(name);
          user.setBounds(130, 50, 180,30 );
          add(user);
          login.setBounds(100, 90, 60, 30 );
          add(login);
  }
  public static void main(String[] args)
        {
      Frame sb=new Setbound("SetBound");
```

```
        sb.setSize(350,150);
        sb.setVisible(true);
    }
}
```

Output:

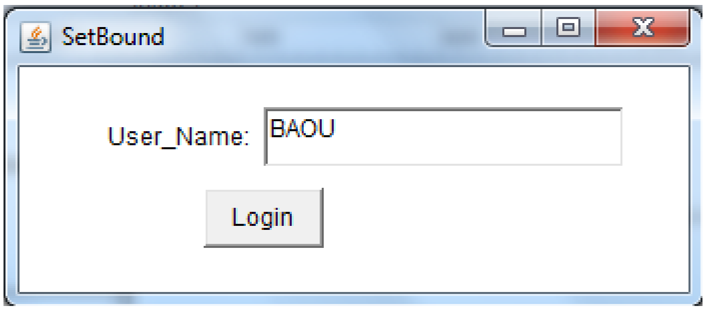


Figure-124: Output of program

➢ **Check Your Progress 3**

1) What is the function of a LayoutManager in Java?

……………………………………………………………………………………

……………………………………………………………………………………

2) Why do you want to use a null layout manager?

……………………………………………………………………………………

……………………………………………………………………………………

3) Which method will cause a Frame to be displayed?

……………………………………………………………………………………

……………………………………………………………………………………

4) Write the advantages of layout manager over traditional windowing systems.

……………………………………………………………………………………

……………………………………………………………………………………

5) How the elements of a CardLayout are organized?

…………………………………………………………………………………

…………………………………………………………………………………

6) What is the difference between GridLayout and GridBagLayout?

…………………………………………………………………………………

…………………………………………………………………………………

3.6 LET US SUM UP

In this unit we have learned the basics of how to paint, including how to use the graphics primitives to draw basic shapes, how to use fonts and font metrics to draw text, and how to use Color objects to change the color of what we are drawing on the container. Graphics, Color and Font classes are the foundation in painting that enables user to do animation inside a container and to work with images. Layout Manager plays a crucial role for arranging components as per the user requirement for designing attractive and user friendly GUI.

3.7 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

➢ **Check Your Progress 1**

1. State information of Graphics class includes he Component object on which to draw, translation origin for rendering and clipping coordinates, current clip, current color, current font, current logical pixel operation function, current XOR alternation color.

2. First task is of the graphics context. The graphics context is information that will affect drawing operations. This includes the background and foreground colors, the font and the location and dimensions of the clipping rectangle. It even includes information about the screen or image. Second role is that, Graphics class provides methods for drawing simple geometric shapes, text and images to the graphics destination. All output to the graphics destination occurs via an invocation of various methods methods.

3. The Color class creates color by using the RGBA values. RGBA stands for RED, GREEN, BLUE, ALPHA. The value for individual components RGBA

ranges from 0 to 255 or 0.0 to 0.1. The value of alpha determines the opacity of the color, where 0 or 0.0 represents fully transparent and 255 or 1.0 represents opaque.

4. A Canvas object enables user to access to a Graphics object via its paint() method.

- **Check Your Progress 2**

1**.** There are four styles for displaying fonts in Java. They are plain, bold, italic and bold italic. Three class constants are used to represent font styles:

   a. public static final int BOLD: This constant represents a boldface font.
   b. public static final int ITALIC: This constant represents an italic font.
   c. public static final int PLAIN: This constant represents a plain or normal font.

2. Paint is called for the first time when the container is loaded. Every Java Component implements paint(Graphics), which is responsible for painting that component in the Graphics context passed as the parameter. When we extend a Component and want to display it differently than its superclass, we have to override public void paint(Graphics) .

   Whereas repaint method is called everytime the container is refreshed. The repaint() method is sent to a Component when it needs to be repainted. For example, a window is moved or resized or unhidden. It also happens when a webpage contains an image and the pixels of the image are arriving slowly. The action of repaint() is to spawn a new Thread, which schedules update(Graphics) in 100 milliseconds. If another repaint() happens before the 100 milliseconds time, the previous update() is cancelled and a new one is scheduled.

3. Various method of Font class is described in following table.

Method Name	**Object**	**Description**
getFont()	Graphics	It will return the current font object as previously set by setFont()
getName()	Font	It will return the name of the font as a string
getSize()	Font	It will return the current font size (an integer)
getStyle()	Font	It will return the current style of the font (styles are integer constants: 0 is plain, 1 is bold, 2 is italic, 3 is bold italic)
isPlain()	Font	It will return true or false if the font's style is plain
isBold()	Font	It will return true or false if the font's style is bold
isItalic()	Font	It will return true or false if the font's style is italic

Table-11: Methods of Font Class

4. The FontMetrics class is used to define implementation-specific properties such as ascent and descent, of a Font object.

- **Check Your Progress 3**

1. A LayoutManager implements some policy for arranging components added to a container. It sets the sizes and positions of the components. Different layout managers have different rules for arranging components. The standard layout manager classes are BorderLayout, GridLayout etc.

2. If the layout manager for a container is set to null, then the programmer has to set the sizes and positions of all the components in the container. This

gives the programmer more flexibility over the layout. For simple layouts that does not change size in a container, the setBounds() method of each component will be called when it is added to the container. When the container can change size, then the sizes and positions should be recomputed whenever a change in size occurs. This task is performed by a layout manager automatically, and due to this it is good to use a layout manager for a container that can change size.

3. show() and setVisible() method
4. Java uses layout managers to layout components in a consistent manner across all windowing platforms. Java's layout managers are not bind to absolute sizing and positioning, they can accomodate platform-specific differences among windowing systems.
5. The elements of a CardLayout are stacked, one upon other like a deck of cards.
6. In Grid layout the size of each grid remains constant while in GridbagLayout grid size can be varied.

3.8 FURTHER READING

11) Core Java Programming-A Practical Approach by Tushar B. Kute
12) Java: The Complete Reference by Schildt Herbert. Ninth Edition
13) Head First Java: A Brain-Friendly Guide, Kindle Edition by Kathy Sierra, Bert Bates. 2nd Edition
14) Java: A Beginner’s Guide by Schildt Herbert Sixth Edition
15) Core Java Volume I — Fundamentals by Cay S. HorstMann, Gary Cornell, 9th Edition
16) https://www.codemiles.com/java-examples/fonts-in-java-t2831.html
17) https://courses.cs.washington.edu/courses/cse3151/98au/java/jdk1.2beta15/docs/api/java/awt/Font.html
18) https://www.leepoint.net/GUI-appearance/fonts/10font.html

3.9 ASSIGNMENTS

9) Define Graphics. Explain the importance of Graphics class in java.
10) Differentiate paint(), repaint() and update() method.
11) Explain Font class with proper example to demonstrate the use of font family.
12) How do we can set and get color in java application? Explain through example.
13) What is Layout Manager? Explain different types of layout managers.