Graphical User Interface

HADAIQ ROLIS SANABILA

Python GUI

- There are many GUI (Graphical User Interface) modules available for developing GUI programs in Python.
  - Tkinter
  - wxWidgets
  - Qt
  - Gtk+
  - FLTK
  - FOX
  - OpenGL
- This chapter introduces Tkinter, which will enable you to develop GUI projects.
- Tkinter is not only a useful tool for developing GUI projects, but it is also a valuable pedagogical tool for learning object-oriented programming

Tkinter

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Tkinter (pronounced T-K-Inter) is short for "Tk interface." Tk is a GUI library used by many programming languages for developing GUI programs.
- Tkinter provides an interface for Python programmers to use the Tk GUI library, and it is the de-facto standard for developing GUI programs in Python

Tkinter

- The **tkinter** module contains the classes for creating GUIs.
- The **Tk** class creates a window for holding GUI widgets (i.e., visual components).

```
# ProcessButtonEventClass.py
from Tkinter import * # Import all definitions from tkinter
class ProcessButtonEvent:
    def __init__(self):
        window = Tk() # Create a window
        label = Label(window, text = "Welcome to Python")
        btOK = Button(window, text = "OK", fg = "red",command = self.processOK )
        btCancel = Button(window, text = "Cancel", bg = "yellow", command = self.processCancel )
        label.pack() # Place the label in the window
        btOK.pack() # Place the OK button in the window
        btCancel.pack() # Place the Cancel button in the window
        window.mainloop() # Create an event loop
    def processOK(self):
        print("OK button is clicked")
    def processCancel(self):
        print("Cancel button is clicked")
ob = ProcessButtonEvent() # invoke __init__ method
```

Tkinter

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Whenever you create a GUI-based program in Tkinter, you need to import the **tkinter** module and create a window by using the **Tk** class.
- The asterisk (*****) in the import statement indicates that it imports all definitions for classes, functions, and constants from the **tkinter** module to the program.
- **Tk()** creates an instance of a window.
- **Label** and **Button** are Python Tkinter *widget classes* for creating labels and buttons.
- The first argument of a widget constructor is always the *parent container* (i.e., the container in which the widget will be placed)

Widget on Window

- The statement

  ```
  label = Label(window, text = "Welcome to Python")
  ```

  constructs a label with the text Welcome to Python that is contained in the window.
- The statement

  label.pack()

  places **label** in the container using a pack manager.
- In this example, the pack manager packs the widget in the window row by row. More on the pack manager will be introduced later.

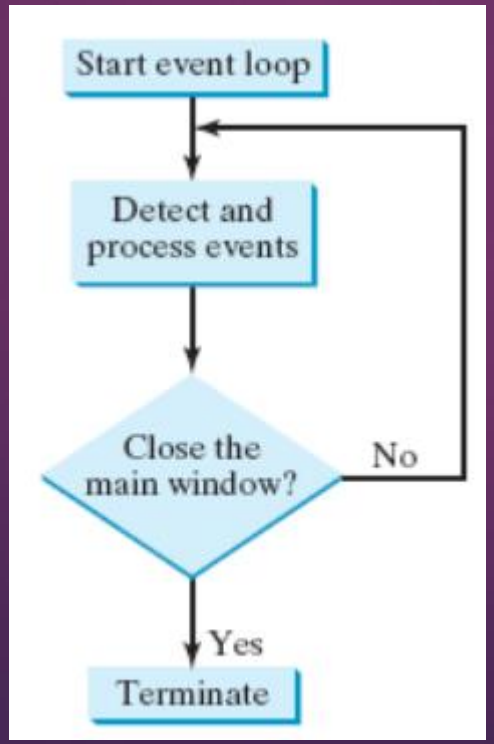
Event Processing

- Tkinter GUI programming is event driven. After the user interface is displayed, the program waits for user interactions such as mouse clicks and key presses.
- This is specified in the statement `window.mainloop()`
- The statement creates an event loop. The event loop processes events continuously until you close the main window

Event Processing

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- A Tkinter GUI program listens and processes events in a continuous loop



Event Processing

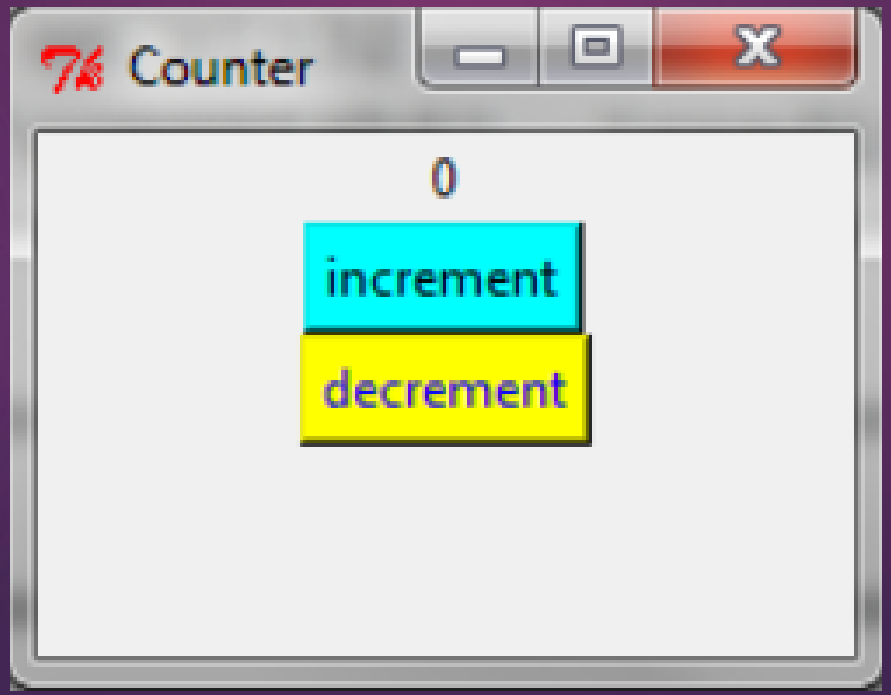
- A Tkinter widget can be bound to a function or method, which is called when an event occurs.
- For example:
  - When the user clicks a button, your program should process this event.
  - You enable this action by defining a processing function and binding the function to the button

GUI Class

- There are two advantages of defining a class for creating a GUI and processing GUI events.
  - First, you can reuse the class in the future.
  - Second, defining all the functions as methods enables them to access instance data fields in the class.

Exercise

- Write a GUI-based program that displays a count and two buttons: one for incrementing the count and the other for decrementing the count.



Widget Classes

Widget Class	Description
Button	A simple button, used to execute a command.
Canvas	An area to display graphical elements like lines or text.
Checkbutton	Clicking a check button toggles between the values.
Entry	A text entry field, also called a text field or a text box.
Frame	A container widget for containing other widgets.
Label	Displays text or an image.
Menu	A menu pane, used to implement pull-down and popup menus.
Menubutton	A menu button, used to implement pull-down menus.
Message	Displays a text. Similar to the label widget, but can automatically wrap text to a given width or aspect ratio.
Radiobutton	Clicking a radio button sets the variable to that value, and clears all other radio buttons associated with the same variable.
Text	Formatted text display. Allows you to display and edit text with various styles and attributes.

Widget Class

- There are many options for creating widgets from these classes. The first argument is always the parent container.
- You can specify a foreground color, background color, font, and cursor style when constructing a widget.

Color

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- To specify a color, use either a color name (such as red, yellow, green, blue, white, black, purple)
- or explicitly specify the red, green, and blue (RGB) color components by using a string **#RRGGBB**, where **RR**, **GG**, and **BB** are hexadecimal representations of the red, green, and blue values, respectively.

Font

- You can specify a font in a string that includes the font name, size, and style.
- Examples:
  Times 10 bold
  Helvetica 10 bold italic
  CourierNew 20 bold italic
  Courier 20 bold italic underline

Text Formatting

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- By default, the text in a label or a button is centered.
- You can change its alignment by using the **justify** option with the named constants **LEFT**, **CENTER**, or **RIGHT**.
- You can also display the text in multiple lines by inserting the newline character.

Changing Widget Properties

- When you construct a widget, you can specify its properties such as fg, bg, font, cursor, text, and command in the constructor. Later in th program, you can change the widget's properties by using the following syntax:

```
widgetName["propertyName"] = newPropertyValue
```

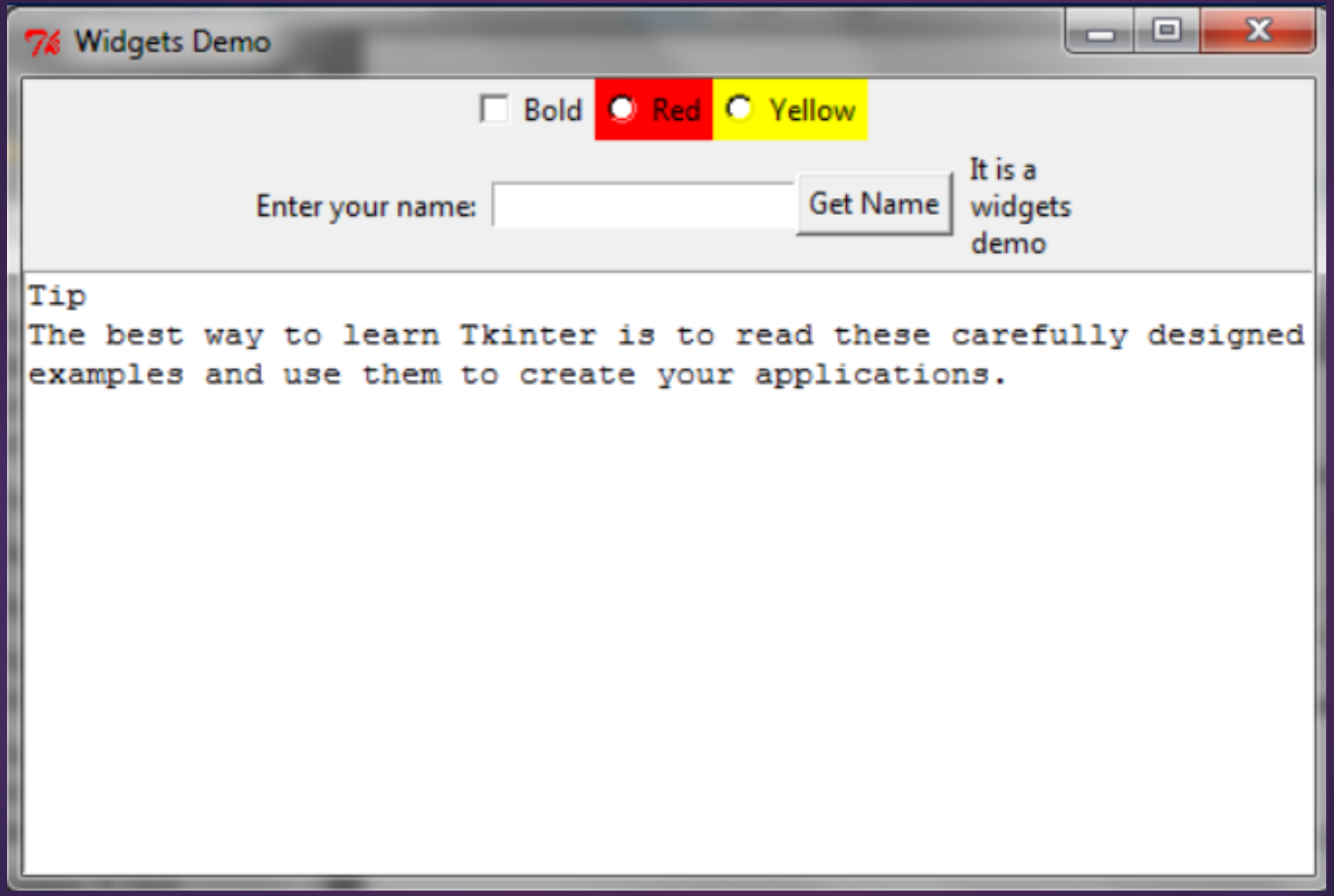
- For example, the following code creates a button and its properties are changed.

```
btShowOrHide = Button(window,text= "Show",bg= "white")
btShowOrHide["text"] = "Hide"
btShowOrHide["bg"] = "red"
btShowOrHide["fg"] = "#AB84F9"
btShowOrHide["cursor"] = "plus"
btShowOrHide["justify"] = LEFT
```

```
from Tkinter import *
class WidgetsDemo:
    def __init__(self):
        window = Tk() # Create a window
        window.title("Widgets Demo") # Set a title
        # Create and add a frame to window
        frame1 = Frame(window)
        frame1.pack()
        # Add a button, a check button, and a radio button to frame1
        self.v1 = IntVar()
        cbtBold = Checkbutton(frame1, text = "Bold", variable = self.v1, command = self.processCheckbutton)
        self.v2 = IntVar()
        rbRed = Radiobutton(frame1, text = "Red", bg = "red", variable = self.v2, value = 1,  command =
self.processRadiobutton)
        rbYellow = Radiobutton(frame1, text = "Yellow",bg = "yellow", variable = self.v2, value = 2,command =
self.processRadiobutton)
        cbtBold.grid(row = 1, column = 1)
        rbRed.grid(row = 1, column = 2)
        rbYellow.grid(row = 1, column = 3)
```

```
# Add a label, an entry, a button and a message to frame2
frame2 = Frame(window) # Create and add a frame to window
frame2.pack()
label = Label(frame2, text = "Enter your name: ")
self.name = StringVar()
entryName = Entry(frame2, textvariable = self.name)
btGetName = Button(frame2, text = "Get Name", command = self.processButton)
message = Message(frame2, text = "It is a widgets demo")
label.grid(row = 1, column = 1)
entryName.grid(row = 1, column = 2)
btGetName.grid(row = 1, column = 3)
message.grid(row = 1, column = 4)
# Add a text
text = Text(window) # Create a text, add to the window
text.pack()
text.insert(END,"Tip\nThe best way to learn Tkinter is to read ")
text.insert(END,"these carefully designed examples and use them ")
text.insert(END, "to create your applications.")
window.mainloop() # Create an event loop
```

```
    def processCheckbutton(self):
        print("check button is " + ("checked " if self.v1.get() == 1 else "unchecked"))
    def processRadiobutton(self):
        print(("Red" if self.v2.get() == 1 else "Yellow") + " is selected " )
    def processButton(self):
        print("Your name is " + self.name.get())

WidgetsDemo() # Create GUI
```



Changing Widget Properties

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- You use an entry (text field) for entering a value. The value just be an object of **IntVar**, **DoubleVar**, or **StringVar** representing an integer, a float, or a string, respectively.
- **IntVar**, **DoubleVar**, and **StringVar** are defined in the **tkinter** module.
- The program creates a check button and associates it with the variable **v1**. **v1** is an instance of **IntVar**. **v1** is set to **1** if the check button is checked, or **0** if it isn't checked. When the check button is clicked, Python invokes the **processCheckbutton** method.

Changing Widget Properties

- The grid *geometry manager* is used to place the check button and radio buttons into **frame1**. These three widgets are placed in the same row and in columns 1, 2, and 3, espectively.
- An entry is created and associated with the variable **name** of the **StringVar** type for storing the value in the entry.
- When you click the *Get Name* button, the **processButton** method displays the value in the entry.
- The **Message** widget is like a label except that it automatically wraps the words and displays them in multiple lines.
- The program also creates a **Text** widget for displaying and editing text. You can use the **insert** method to insert text into this widget. The **END** option specifies that the text is inserted into the end of the current content.

Canvas

- You use the **Canvas** widget for displaying shapes.
- You can use the methods
  - **create_rectangle**
  - **create_oval**
  - **create_arc**
  - **create_polygon**
  - **create_line**

Canvas

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

```
from Tkinter import *
class CanvasDemo:
    def __init__(self):
        window = Tk() # Create a window
        window.title("Canvas Demo") # Set title
        # Place self.canvas in the window
        self.canvas = Canvas(window, width = 200, height = 100, bg = "white")
        self.canvas.pack()
        # Place buttons in frame
        frame = Frame(window)
        frame.pack()
```

Canvas

```
btRectangle = Button(frame, text = "Rectangle", command = self.displayRect)
btOval = Button(frame, text = "Oval",  command = self.displayOval)
btArc = Button(frame, text = "Arc", command = self.displayArc)
btPolygon = Button(frame, text = "Polygon",  command = self.displayPolygon)
btLine = Button(frame, text = "Line",  command = self.displayLine)
btString = Button(frame, text = "String",  command = self.displayString)
btClear = Button(frame, text = "Clear",    command = self.clearCanvas)
btRectangle.grid(row = 1, column = 1)
btOval.grid(row = 1, column = 2)
btArc.grid(row = 1, column = 3)
btPolygon.grid(row = 1, column = 4)
btLine.grid(row = 1, column = 5)
btString.grid(row = 1, column = 6)
btClear.grid(row = 1, column = 7)
window.mainloop() # Create an event loop
```

Canvas

```
    def displayRect(self):
        self.canvas.create_rectangle(10, 10, 190, 90, tags = "rect")
        # Display an oval
    def displayOval(self):
        self.canvas.create_oval(10, 10, 190, 90, fill = "red",tags = "oval")
        # Display an arc
    def displayArc(self):
        self.canvas.create_arc(10, 10, 190, 90, start = 0, extent = 90, width = 8, fill =
"red", tags ="arc")
    # Display a polygon
    def displayPolygon(self):
        self.canvas.create_polygon(10, 10, 190, 90, 30, 50,tags = "polygon")
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

Canvas

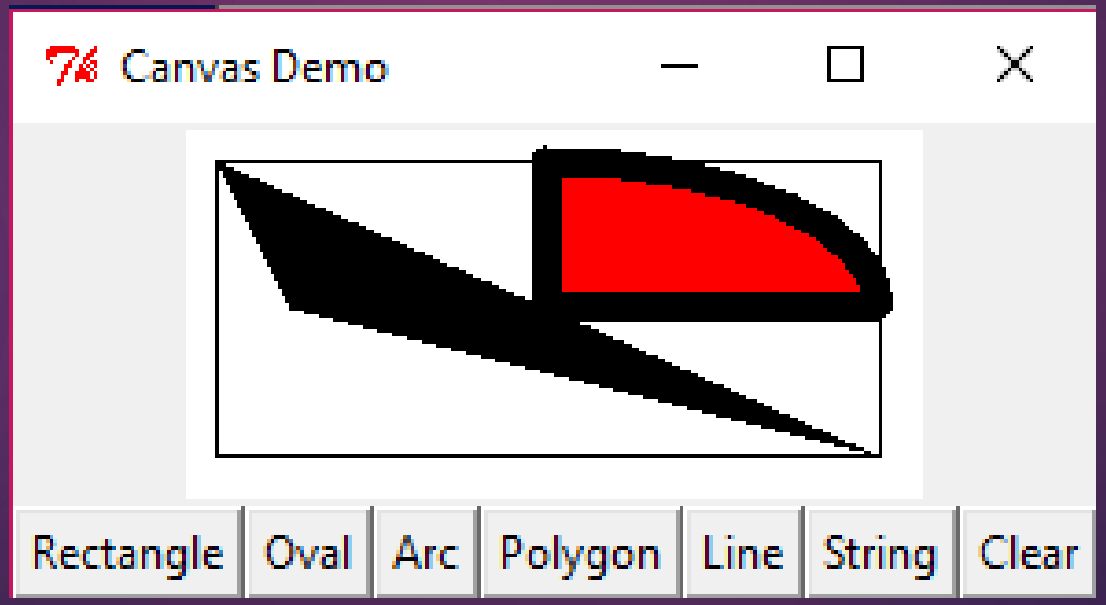
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

```
# Display a line

    def displayLine(self):

        self.canvas.create_line(10, 10, 190, 90, fill = "red",tags = "line")

        self.canvas.create_line(10, 90, 190, 10, width = 9, arrow = "last",
activefill = "blue", tags = "line")

# Display a string

    def displayString(self):

        self.canvas.create_text(60, 40, text = "Hi, I am a string", font = "Times
10 bold underline", tags = "string")

# Clear drawings

    def clearCanvas(self):

        self.canvas.delete("rect", "oval", "arc", "polygon","line", "string")


CanvasDemo() # Create GUI
```

Canvas

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Seven buttons—labeled with the text *Rectangle*, *Oval*, *Arc*, *Polygon*, *Line*, *String*, and *Clear*—are created.
- The *grid manager* places the buttons in one row in a frame



Coordinate System

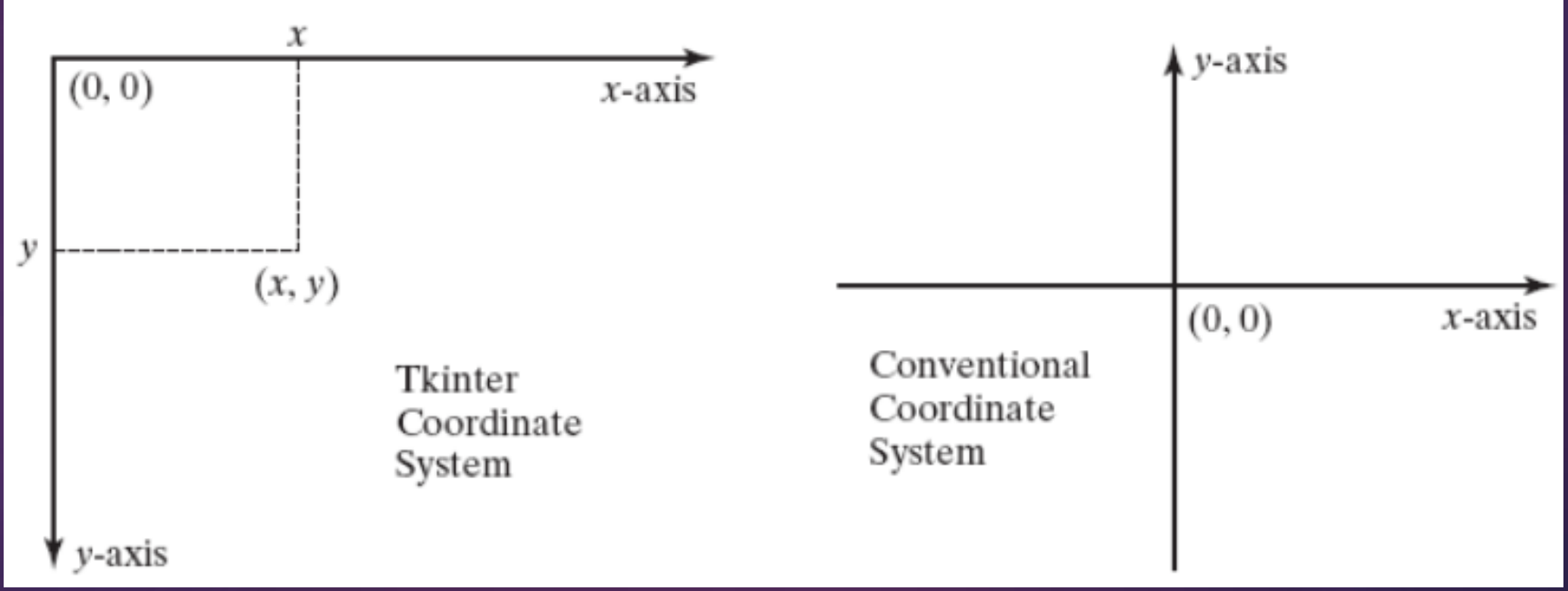
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- To draw graphics, you need to tell the widget where to draw.
- Each widget has its own coordinate system with the origin (**0**, **0**) at the upper-left corner.
- The x-coordinate increases to the right, and the y-coordinate increases downward.
- Note that the Tkinter coordinate system differs from the conventional coordinate system.

Coordinate System

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

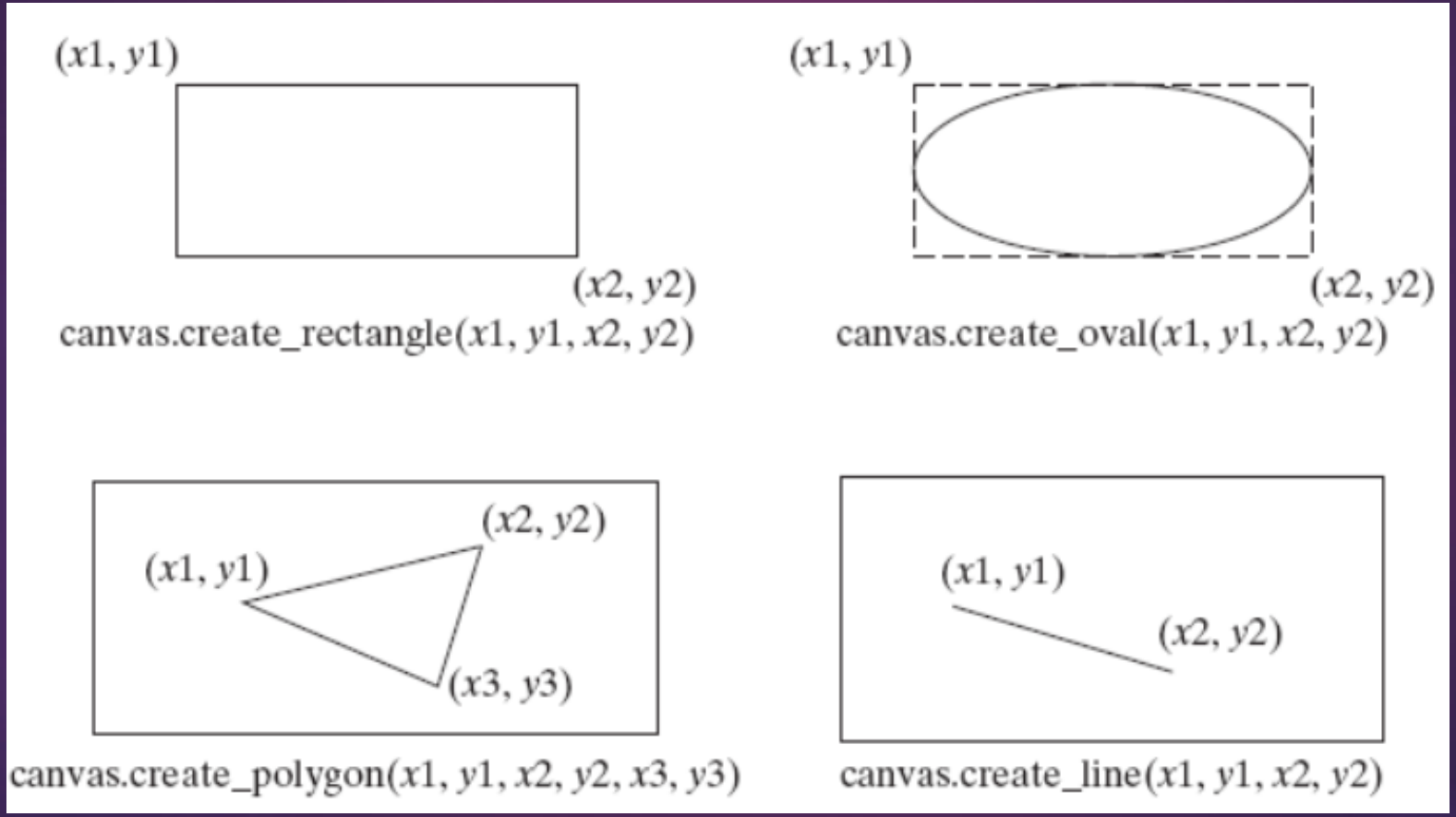
- The Tkinter coordinate system is measured in pixels, with **(0, 0)** at its upperleft corner.



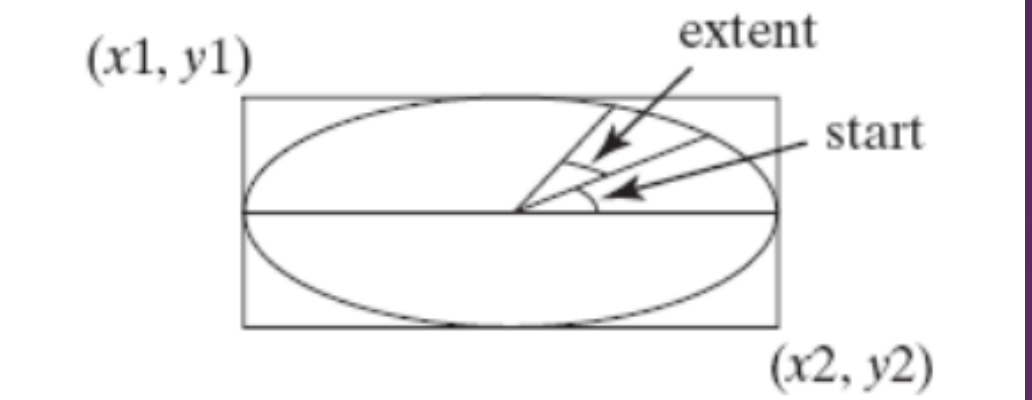
Coordinate System in Canvas

- The methods **create_rectangle**, **create_oval**, **create_arc**, **create_polygon**, and **create_line** are used to draw rectangles, ovals, arcs, polygons, and lines.
- The **create_text** method is used to draw a text string. Note that the horizontal and vertical center of the text is displayed at (x, y) for **create_text(x, y, text).**
- All the drawing methods use the **tags** argument to identify the drawing. These tags are used the **delete** method for clearing the drawing from the canvas

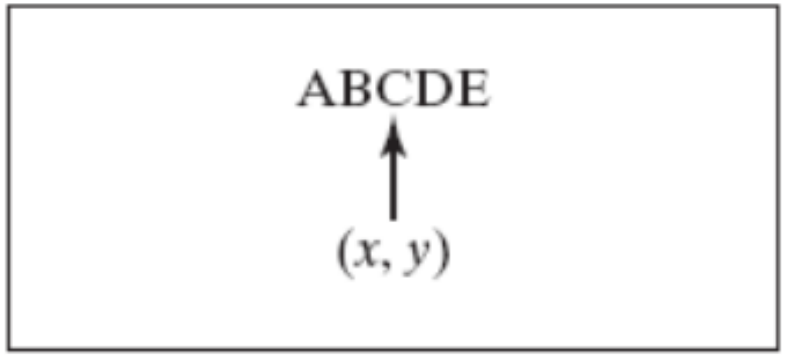
Canvas


(x1, y1)
(x2, y2)
canvas.create_rectangle(x1, y1, x2, y2)
(x1, y1)
(x2, y2)
canvas.create_oval(x1, y1, x2, y2)
(x2, y2)
(x1, y1)
(x3, y3)
canvas.create_polygon(x1, y1, x2, y2, x3, y3)
(x1, y1)
(x2, y2)
canvas.create_line(x1, y1, x2, y2)

Canvas



canvas.create_arc(x1, y1, x2, y2, start, extent)



canvas.create_text(x, y, text = "ABCDE")

Canvas

- The **width** argument can be used to specify the pen size in pixels for drawing the shapes.
- The **arrow** argument can be used with **create_line** to draw a line with an arrowhead. The arrowhead can appear at the start, end, or both ends of the line with the argument value **first**, **end**, or **both**.
- The **activefill** argument makes the shape change color when you move the mouse cursor over it

Geometry Manager

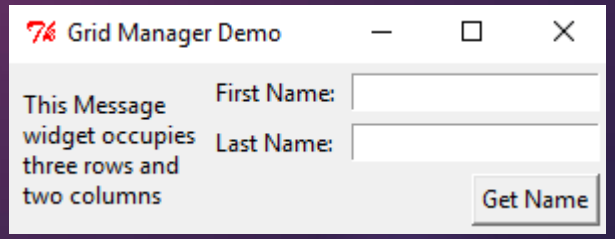
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Tkinter uses a geometry manager to place widgets inside a container.
- Tkinter supports three geometry managers
  - grid manager
  - pack manager
  - place manager
- Since each manager has its own style of placing the widget, it is not a good practice to mix the managers for the widgets in the same container.
- You can use a frame as a subcontainer to achieve the desired layout

Geometry Manager

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- The grid manager places widgets into the cells of an invisible grid in a container.
- You can place a widget in a specified row and column.
- You can also use the **rowspan** and **columnspan** parameters to place a widget in multiple rows and columns.
- Study the following example



Geometry Manager

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

```
#GridManagerDemo.py
from Tkinter import *
class GridManagerDemo:
    window = Tk() # Create a window
    window.title("Grid Manager Demo") # Set title
    message = Message(window, text = "This Message widget occupies three rows and two columns")
    message.grid(row = 1, column = 1, rowspan = 3, columnspan = 2)
    Label(window, text = "First Name:").grid(row = 1, column = 3)
    Entry(window).grid(row = 1, column = 4, padx = 5, pady = 5)
    Label(window, text = "Last Name:").grid(row = 2, column = 3)
    Entry(window).grid(row = 2, column = 4)
    Button(window, text = "Get Name").grid(row = 3, padx = 5, pady = 5, column = 4, sticky = E)
    window.mainloop() # Create an event loop
GridManagerDemo() # Create GUI
```

Geometry Manager

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- The **Message** widget is placed in row 1 and column 1 and it expands to three rows and two columns.
- The *Get Name* button uses the **sticky = E** option to stick to the east in the cell so that it is right aligned with the **Entry** widgets in the same column.
- The **sticky** option defines how to expand the widget if the resulting cell is larger than the widget itself.
- The **sticky** option can be any combination of the named constants **S**, **N**, **E**, and **W**, or **NW**, **NE**, **SW**, and **SE**.
- The **padx** and **pady** options pad the optional horizontal and vertical space in a cell. You can also use the **ipadx** and **ipady** options to pad the optional horizontal and vertical space inside the widget borders

Displaying Images

- You can add an image to a label, button, check button, or radio button.
- To create an image, use the **PhotoImage** class as follows:

```
photo = PhotoImage(file = imagefilename)
```

- The image file must be in GIF format. You can use a conversion utility to convert image files in other formats into GIF format.
- You can also use the **create_image** method to display an image in a canvas. In fact, you can display multiple images in one canvas.
- Study the following examples

Displaying Image

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

```
from Tkinter import *
class ImageDemo:
    def __init__(self):
        window = Tk() # Create a window
        window.title("Image Demo") # Set title
        # Create PhotoImage objects
        pinkImage = PhotoImage(file = "image/pink.gif")
        pinkImage  = pinkImage .subsample(2,2)
        flowerImage = PhotoImage(file = "image/flower.gif")
        flowerImage  = flowerImage .subsample(3,3)
        leftImage = PhotoImage(file = "image/left.gif")
        rightImage = PhotoImage(file = "image/right.gif")
        gerImage = PhotoImage(file = "image/germany.gif")
        ukImage = PhotoImage(file = "image/uk.gif")
        crossImage = PhotoImage(file = "image/cross.gif")
        circleImage = PhotoImage(file = "image/circle.gif")
```

Displaying Image

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

```
# frame1 to contain label and canvas
frame1 = Frame(window)
frame1.pack()
Label(frame1, image = flowerImage).pack(side = LEFT)
canvas = Canvas(frame1)
canvas.create_image(150, 100, image = pinkImage)
canvas["width"] = 300
canvas["height"] = 200
canvas.pack(side = LEFT)
```

Displaying Image

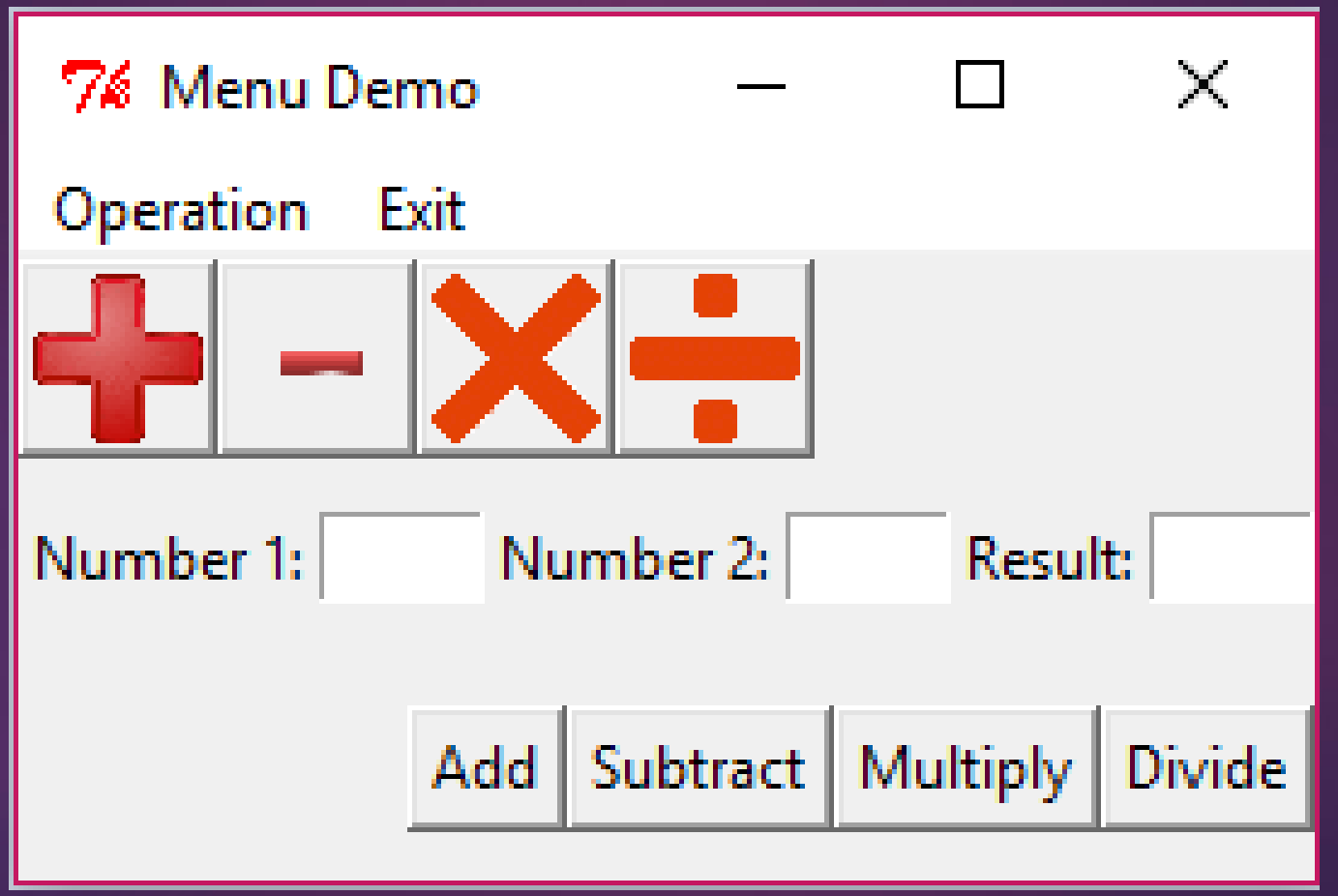
UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

```
        # frame2 to contain buttons, check buttons, and radio buttons
        frame2 = Frame(window)
        frame2.pack()
        radio = IntVar ()
        Button(frame2, image = leftImage).pack(side = LEFT)
        Button(frame2, image = rightImage).pack(side = LEFT)
        Checkbutton(frame2, image = gerImage).pack(side = LEFT)
        Checkbutton(frame2, image = ukImage).pack(side = LEFT)
        Radiobutton(frame2, image = crossImage, variable = radio,value = 1).pack(side = LEFT)
        Radiobutton(frame2, image = circleImage, variable = radio,value = 2).pack(side = LEFT)
        window.mainloop() # Create an event loop
ImageDemo()
```

Menus

- You can use Tkinter to create menus, popup menus, and toolbars.
- Menus make selection easier and are widely used in windows.
- You can use the **Menu** class to create a menu bar and a menu, and use the **add_command** method to add items to the menu.
- See the next example

Menu



Menu

```
#GridManagerDemo.py
#MenuDemo.py
from Tkinter import *
class MenuDemo:
    def __init__(self):
        window = Tk()
        window.title("Menu Demo")
        # Create a menu bar
        menubar = Menu(window)
        window.config(menu = menubar) # Display the menu bar
        # create a pulldown menu, and add it to the menu bar
        operationMenu = Menu(menubar, tearoff = 0)
        menubar.add_cascade(label = "Operation", menu = operationMenu)
        operationMenu.add_command(label = "Add", command = self.add)
        operationMenu.add_command(label = "Subtract", command = self.subtract)
```

Menu


```
operationMenu.add_separator()
operationMenu.add_command(label = "Multiply", command = self.multiply)
operationMenu.add_command(label = "Divide", command = self.divide)
# create more pulldown menus
exitmenu = Menu(menubar, tearoff = 0)
menubar.add_cascade(label = "Exit", menu = exitmenu)
exitmenu.add_command(label = "Quit", command = window.quit)
# Add a tool bar frame
frame0 = Frame(window) # Create and add a frame to window
frame0.grid(row = 1, column = 1, sticky = W)
# Create images
plusImage = PhotoImage(file = "image/plus.gif")
minusImage = PhotoImage(file = "image/minus.gif")
timesImage = PhotoImage(file = "image/multiply.gif")
divideImage = PhotoImage(file = "image/divide.gif")
```

Menu

```
Button(frame0, image = plusImage, command = self.add).grid(row = 1, column = 1, sticky = E)
Button(frame0, image = minusImage, command = self.subtract).grid(row = 1, column = 2)
Button(frame0, image = timesImage, command = self.multiply).grid(row = 1, column = 3)
Button(frame0, image = divideImage, command = self.divide).grid(row = 1, column = 4)
# Add labels and entries to frame1
frame1 = Frame(window)
frame1.grid(row = 2, column = 1, pady = 10)
Label(frame1, text = "Number 1:").pack(side = LEFT)
self.v1 = StringVar()
Entry(frame1, width = 5, textvariable = self.v1, justify = RIGHT).pack(side = LEFT)
Label(frame1, text = "Number 2:").pack(side = LEFT)
self.v2 = StringVar()
Entry(frame1, width = 5, textvariable = self.v2, justify = RIGHT).pack(side = LEFT)
Label(frame1, text = "Result:").pack(side = LEFT)
self.v3 = StringVar()
Entry(frame1, width = 5, textvariable = self.v3, justify = RIGHT).pack(side = LEFT)
```

Menu

```
    # Add buttons to frame2
        frame2 = Frame(window) # Create and add a frame to window
        frame2.grid(row = 3, column = 1, pady = 10, sticky = E)
        Button(frame2, text = "Add", command = self.add).pack(side = LEFT)
        Button(frame2, text = "Subtract", command = self.subtract).pack(side = LEFT)
        Button(frame2, text = "Multiply", command = self.multiply).pack(side = LEFT)
        Button(frame2, text = "Divide", command = self.divide).pack(side = LEFT)
        mainloop()
    def add(self):
        self.v3.set(eval(self.v1.get()) + eval(self.v2.get()))
    def subtract(self):
        self.v3.set(eval(self.v1.get()) - eval(self.v2.get()))
    def multiply(self):
        self.v3.set(eval(self.v1.get()) * eval(self.v2.get()))
    def divide(self):
        self.v3.set(eval(self.v1.get()) / eval(self.v2.get()))
MenuDemo() # Create GUI
```

Menus

- The program creates a menu bar, and the menu bar is added to the window.
- To display the menu, use the **config** method to add the menu bar to the container.
- To create a menu inside a menu bar, use the menu bar as the parent container and invoke the menu bar's **add_cascade** method to set the menu label.
- Use the **add_command** method to add items to the menu.
- Note that the **tearoff** is set to **0**, which specifies that the menu cannot be moved out of the window. If this option is not set, the menu can be moved out of the window.
- The program creates another menu named **Exit** and adds the **Quit** menu item to it.

Menus

- The program creates a frame named **frame0** and uses it to hold toolbar buttons.
- The toolbar buttons are buttons with images, which are created by using the **PhotoImage** class. The command for each button specifies a callback function to be invoked when a toolbar button is clicked.
- The program creates a frame named **frame1** and uses it to hold labels and entries for numbers. Variables **v1**, **v2**, and **v3** bind the entries.
- The program creates a frame named **frame2** and uses it to hold four buttons for performing *Add*, *Subtract*, *Multiply*, and *Divide*.
- The operation button, menu item, and tool bar button have the same callback function, which is invoked when any one of them is clicked.

Mouse, Key Events, and Bindings

- We can use the **bind** method to bind mouse and key events to a widget.
- In the preceding example:

```
# Bind popup to canvas
    self.canvas.bind("<Button-3>", self.popup)
```

- we used the widget's **bind** method to bind a mouse event with a callback handler by using the syntax:

```
widget.bind(event, handler)
```

Mouse, Key Events, and Bindings

- If a matching event occurs, the handler is invoked. In the preceding example, the event is **<Button-3>** and the handler function is **popup**.
- The event is a standard Tkinter object, which is automatically created when an event occurs. Every handler has an event as its argument.
- The following example defines the handler using the event as the argument:

```
def popup(self, event):
    self.menu.post(event.x_root, event.y_root)
```

- The event object has a number of properties describing the event pertaining to the event. For example, for a mouse event, the **event** object uses the **x**, **y** properties to capture the current mouse location in pixels.

Common Events

Event	*Description*
<Bi-Motion>	An event occurs when a mouse button is moved while being held down on the widget.
<Button-*i*>	Button-1, Button-2, and Button-3 identify the left, middle, and right buttons. When a mouse button is pressed over the widget, Tkinter automatically grabs the mouse pointer's location. ButtonPressed-*i* is synonymous with Button-*i*.
<ButtonReleased-*i*>	An event occurs when a mouse button is released.
<Double-Button-*i*>	An event occurs when a mouse button is double-clicked.
<Enter>	An event occurs when a mouse pointer enters the widget.
<Key>	An event occurs when a key is pressed.
<Leave>	An event occurs when a mouse pointer leaves the widget.
<Return>	An event occurs when the *Enter* key is pressed. You can bind any key such as *A*, *B*, *Up*, *Down*, *Left*, *Right* in the keyboard with an event.
<Shift+A>	An event occurs when the *Shift+A* keys are pressed. You can combine *Alt*, *Shift*, and *Control* with other keys.
<Triple-Button-*i*>	An event occurs when a mouse button is triple-clicked.

Events Properties

Event Property	Description
char	The character entered from the keyboard for key events.
keycode	The key code (i.e., Unicode) for the key entered from the keyboard for key events.
keysym	The key symbol (i.e., character) for the key entered from the keyboard for key events.
num	The button number (1, 2, 3) indicates which mouse button was clicked.
widget	The widget object that fires this event.
x and y	The current mouse location in the widget in pixels.
x_root and y_root	The current mouse position relative to the upper-left corner of the screen, in pixels.

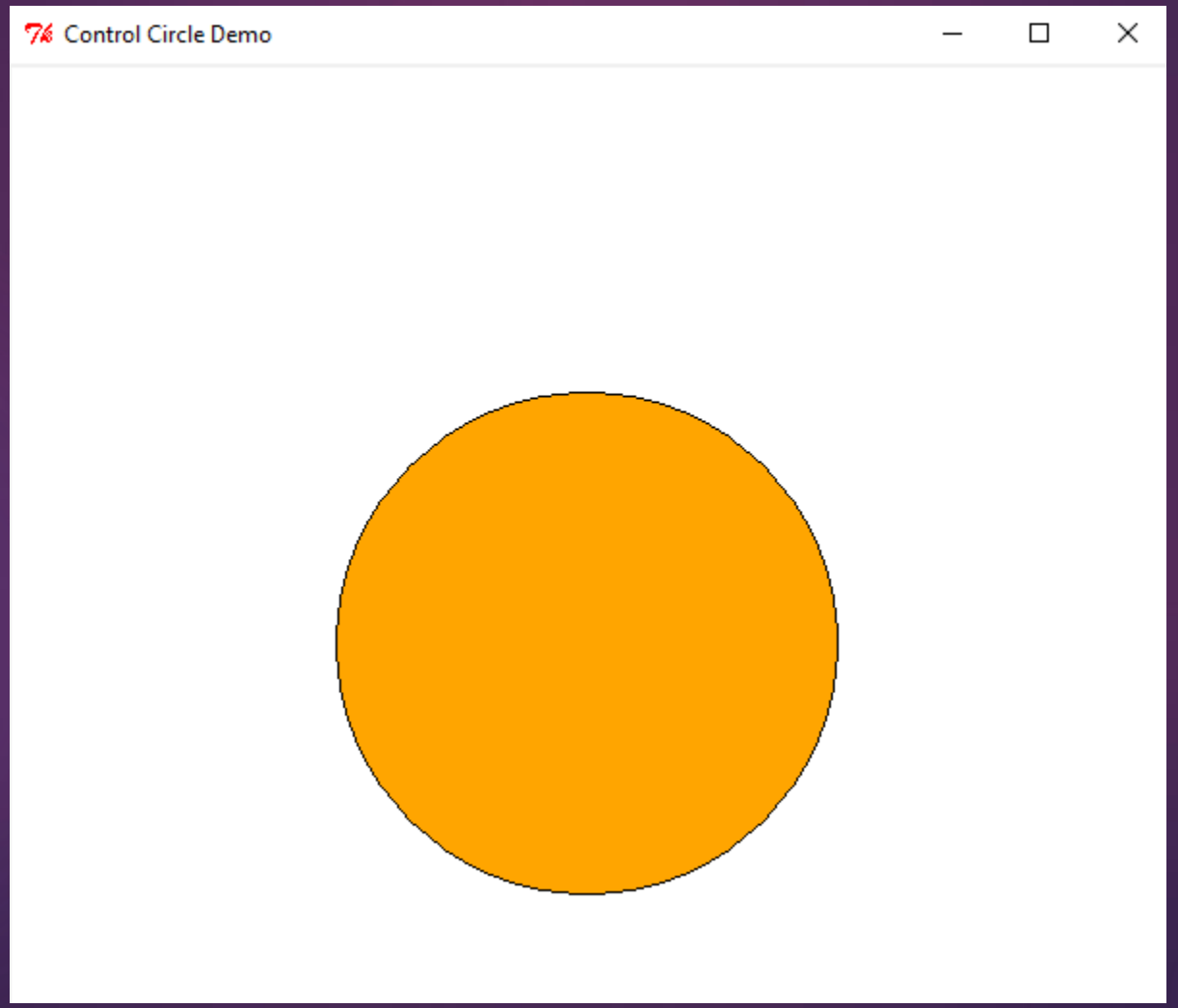
Event

```
#EnlargeShrinkCircle.py
from Tkinter import *
class EnlargeShrinkCircle:
    def __init__(self):
        self.radius = 50
        window = Tk() # Create a window
        window.title("Control Circle Demo") # Set a title
        global w
        global h
        w = 600/2
        h = 600/2
        self.canvas = Canvas(window, bg = "white", width = w*2, height = h*2)
        self.canvas.pack()
        self.canvas.create_oval(w - self.radius, h - self.radius, w + self.radius, h +
self.radius, tags = "oval", fill = "orange")
```

Event

UNIVERSITAS INDONESIA

```
        # Bind canvas with mouse events
        self.canvas.bind("<Button-1>", self.increaseCircle)
        self.canvas.bind("<Button-3>", self.decreaseCircle)
        window.mainloop() # Create an event loop
def increaseCircle(self, event):
        self.canvas.delete("oval")
        self.radius += 20
        self.canvas.create_oval(w- self.radius, h - self.radius, w + self.radius, h + self.radius,
tags = "oval", fill = "orange")
    def decreaseCircle(self, event):
        self.canvas.delete("oval")
        if self.radius > 2:
            self.radius -= 20
            self.canvas.create_oval(w- self.radius, h - self.radius, w + self.radius, h +
self.radius, tags = "oval", fill = "orange")
EnlargeShrinkCircle()
```



UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

Event

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- The program creates a canvas and displays a circle on the canvas with an initial radius of **50**.
- The canvas is bound to a mouse event **<Button-1>** with the handler **increaseCircle** and to a mouse event **<Button-3>** with the handler **decreaseCircle**.
- When the left mouse button is pressed, the **increaseCircle** function is invoked to increase the radius and redisplay the circle .
- When the right mouse button is pressed, the **decreaseCircle** function is invoked tode create the radius and redisplay the circle.

Inheritance and Polymorphism

Inheritance

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Object-oriented programming (OOP) allows you to define new classes from existing classes, through the mechanism of inheritance.
- The procedural paradigm (like in C, Pascal) focuses on designing functions and the object oriented paradigm (like in Python, Java, C++) couples data and methods together into objects.
- The object-oriented approach combines the power of the procedural paradigm with an added dimension that integrates data with operations into objects.

Inheritance

- Inheritance extends the power of the object oriented paradigm by adding an important and powerful feature for reusing software.
- Suppose that you want to define classes to model circles, rectangles, and triangles. These classes have many common features. What is the best way to design these classes to avoid redundancy and make the system easy to comprehend and maintain? The answer is to use inheritance

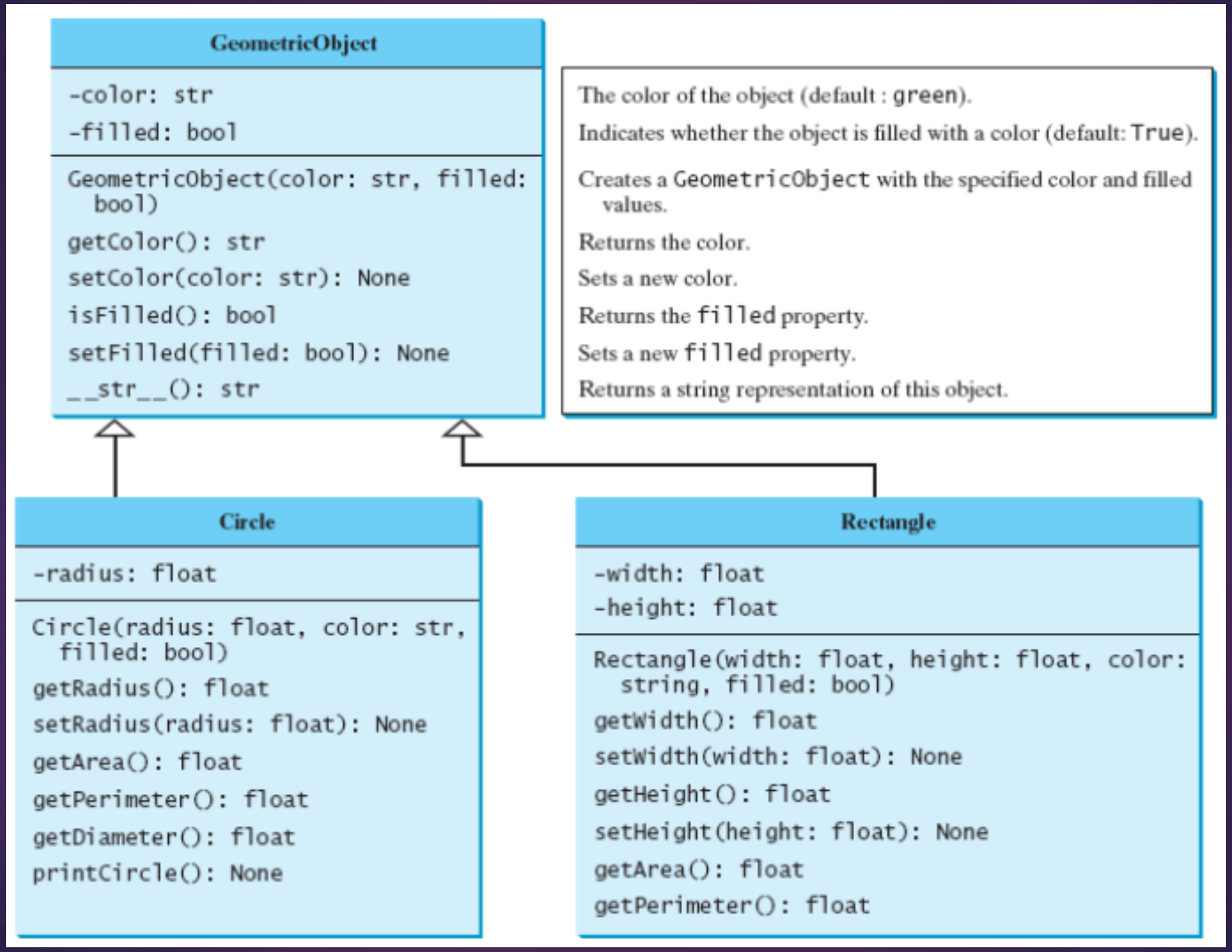
Superclasess and Subclasses

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- You use a class to model objects of the same type.
- Different classes may have some common properties and behaviors that you can generalize in a class, which can then be shared by other classes.
- Inheritance enables you to define a general class (superclass) and later extend it to define more specialized classes (subclasses).
- The specialized classes inherit the properties and methods from the general class.

Superclasess and Subclasses

- In OOP terminology, a class **C1** extended from another class **C2** is called a derived class, child class, or subclass, and **C2** is called a base class, parent class, or superclass.
- In UML, a triangular arrow pointing to the superclass is used to denote the inheritance relationship between the two classes involved



UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

Example

- Because a circle is a special type of geometric object, it shares common properties and methods with other geometric objects. For this reason, it makes sense to define a **Circle** class that extends the **GeometricObject** class.
- Similarly, you can define **Rectangle** as a subclass of **GeometricObject**.
- A subclass inherits accessible data fields and methods from its superclass, but it can also have other data fields and methods.

Example:

```
class GeometricObject:
    def __init__ (self, color = "green", filled = True):
        self.__color = color
        self.__filled = filled
    def getColor (self):
        return self.__color
    def setColor (self, color):
        self.__color = color
    def isFilled    (self):
        return self.__filled
    def setFilled (self, filled):
        self.__filled = filled
    def __str__ (self):
        return "color = "+self.__color + " and filled = "+str(self.__filled)
```

Example:

```
from GeometricObject import GeometricObject
import math
class Circle (GeometricObject):
    def __init__ (self, radius):
        super().__init__()
        self.__radius = radius
    def getRadius (self):
        return self.__radius
    def setRadius (self):
        sekf.__radius = radius
    def getArea(self):
        return self.__radius**2*math.pi
    def getDiamater(self):
        return self.__radius*2
    def getPerimeter(self):
        return self.__radius*2*math.pi
    def printCircle (self):
        print (self.__str__() + ' radius : '+ str(self.__radius))
```

Example:

```
from GeometricObject import GeometricObject
class Rectangle (GeometricObject):
    def __init__ (self, w=1, h=1):
        super().__init__()
        self.__w= w
        self.__h= h
    def getWidth (self):
        return self.__w
    def setWidth (self, w):
        self.__w= w
    def getHeight(self):
        return self.__h
    def setHeight (self, h):
        self.__h= h
    def getArea(self):
        return self.__w*self.__h
    def getPerimeter(self):
        return 2*(self.__w+self.__h)
    def printRect(self):
        print (self.__str__ + ' width: '+ str(self.__w) + ' height: '+str(self.__h))
```

Example

- In our example:
  - The **Circle** class inherits all accessible data fields and methods from the **GeometricObject** class.
  - In addition, it has a new data field, **radius**, and its associated **get** and **set** methods. It also contains the **getArea()**, g**etPerimeter()**, and **getDiameter()** methods for returning the area, perimeter, and diameter of a circle. The **printCircle()** method is defined to print the information about the circle.
  - The **Rectangle** class inherits all accessible data fields and methods from the **GeometricObject** class. In addition, it has the data fields **width** and **height** and the associated **get** and **set** methods.
  - It also contains the **getArea()** and **getPerimeter()** methods for returning the area and perimeter of the rectangle.
- Notes: The get methods are called accessors (--> to access) and the set methods are called mutators (--> to mutate)

- The **Circle** class is derived from the **GeometricObject** class, based on the following syntax:

- This tells Python that the **Circle** class inherits the **GeometricObject** class, thus inheriting the methods **getColor**, **setColor**, **isFilled**, **setFilled**, and **__str__**.
- The **printCircle** method invokes the **__str__()** method to obtain properties defined in the superclass.
- **super().__init__()** calls the superclass's **__init__** method. This is necessary to create data fields defined in the superclass.

Is-a relationship

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Inheritance models the is-a relationships.
- For example:
  - A circle is a geometric object.
  - A car is a vehicle
  - A student is a person
  - An apple is a fruit
  - A cheetah is an animal

Multiple Inheritance

- Python allows you to derive a subclass from several classes. This capability is known as *multiple inheritance*.
- To define a class derived from multiple classes, use the following syntax:

```
class Subclass(SuperClass1, SuperClass2, ...):
initializer
methods
```

Overriding Methods

- A subclass inherits methods from a superclass. Sometimes it is necessary for the subclass to modify the implementation of a method defined in the superclass. This is referred to as method overriding.
- To override a method, the method must be defined in the subclass using the same header as in its superclass.
- Example:
  - The **__str__** method in the **GeometricObject** class returns the string describing a geometric object. This method can be overridden to return the string describing a circle. To override it, add the following new method in Circle.py

Override method __str__ in Circle and Rectangle

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

```
class Circle(GeometricObject):
# Other methods are omitted
# Override the __str__ method defined in
# GeometricObject
def __str__(self):
    return super().__str__() + " radius: " + str(self.__radius)
```

```
class Rectangle(GeometricObject):
# Other methods are omitted
# Override the __str__ method defined in
# GeometricObject
def __str__(self):
    return super().__str__() + " width: " + str(self.__width) + "
height: " + str(self.__height)
```

Private Method

- You can define a private method in Python by adding two underscores in front of a method name.
- A private method cannot be overridden.
- If a method defined in a subclass is private in its superclass, the two methods are completely unrelated, even though they have the same name.

The object class

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Every class in Python is descended from the **object** class.
- The **object** class is defined in the Python library.
- If no inheritance is specified when a class is defined, its superclass is **object** by default.
- For example, the following two class definitions are the same

```
class ClassName:
    ...
```

Equivalent

```
class ClassName(object):
    ...
```

The object class

- It is important to be familiar with the methods provided by the **object** class so that you can use them in your classes.
- All methods defined in the **object** class are special methods with two leading underscores and two trailing underscores.

Polymorphism and Dynamic Binding

- Polymorphism means that an object of a subclass can be passed to a parameter of a superclass type.
- A method may be implemented in several classes along the inheritance chain. Python decides which method is invoked at runtime. This is known as dynamic binding
- A subclass is a specialization of its superclass; every instance of a subclass is also an instance of its superclass, but not vice versa.
- For example, every circle is a geometric object, but not every geometric object is a circle.
- Therefore, you can always pass an instance of a subclass to a parameter of its superclass type

Example:

```
from Circle import Circle
from Rectangle import Rectangle

def displayObject (g):
    print (g.__str__())
def isSameArea (g1,g2):
    return g1.getArea() == g2.getArea()
c = Circle (2)
r = Rectangle (3,2)
displayObject (c)
displayObject (r)
print ("Circle and Rectangle has a same Area ? : ", isSameArea(c,r))
```

```
color = green and filled = True
color = green and filled = True
Circle and Rectangle has a same Area ? :  False
```

Polymorphism

- The **displayObject** method takes a parameter of the **GeometricObject** type.
- You can invoke **displayObject** by passing any instance of **GeometricObject** (for example, **Circle(4)** and **Rectangle(1, 3)**).
- An object of a subclass can be used wherever its superclass object is used. This is commonly known as *polymorphism* (from a Greek word meaning "many forms").

Dynamic Binding

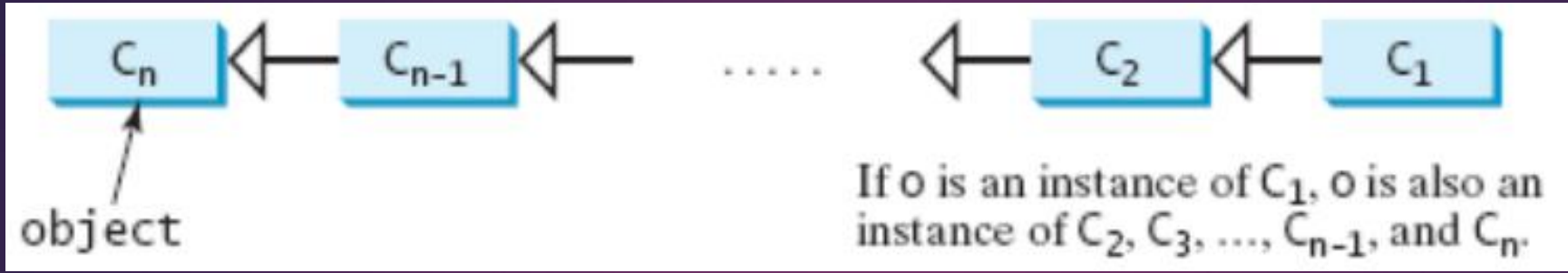
- In the example, c is an object of the Circle class. Circle is a subclass of GeometricObject
- The `__str__()` method is defined in both classes. So which `__str__()` method is invoked by g in the `displayObject`?
- The `__str__()` method invoked by g is determined using dynamic binding

Dynamic Binding

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Dynamic binding works as follows:
  - Suppose an object **o** is an instance of classes $\mathbf{C_1}$, $\mathbf{C_2}$, ..., $\mathbf{C_{n-1}}$, and $\mathbf{C_n}$, where $\mathbf{C_1}$ is a subclass of $\mathbf{C_2}$, $\mathbf{C_2}$ is a subclass of $\mathbf{C_3}$, ..., and $\mathbf{C_{n-1}}$ is a subclass of $\mathbf{C_n}$. That is, $\mathbf{C_n}$ is the most general class, and **C1** is the most specific class. In Python, $\mathbf{C_n}$ is the **object** class.
  - If **o** invokes a method **p**, Python searches the implementation for the method **p** in $\mathbf{C_1}$, $\mathbf{C_2}$, ..., $\mathbf{C_{n-1}}$, and $\mathbf{C_n}$, in this order, until it is found.
  - Once an implementation is found, the search stops and the first-found implementation is invoked.

Dynamic Binding




```
Class Student:
    def __str__(self):
        return "Student"
    def printStudent(self):
        print (self.__str__())
Class GraduateStudent(Student):
    def __str__ (self):
        return "Graduate Student"
a = Student()
b = GraduateStudent()
a.printStudent()
b.printStudent()
```

```
Student
Graduate Student
```

The isinstance function

- The **isinstance** function can be used to determine whether an object is an instance of a class. Its syntax:

```
isinstance(object, ClassName)
```

- Example

```
>>> isinstance("abc",str)
True
>>> isinstance(3,int)
True
>>> isinstance(3,str)
False
>>> from lab09drawShape import Circle
>>> c1 = Circle()
>>> isinstance(c1, Circle)
True
```

The isinstance function

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- Suppose you want to modify the **displayObject** function from the program Test.py to perform the following tasks:
  - Display the area and perimeter of a **GeometricObject** instance.
  - Display the diameter if the instance is a **Circle**, and the width and height if the instance is a **Rectangle**.
- Consider the following example. Invoking **displayObject(c)** passes **c** to **g**. **g** is now an instance of **Circle**. The program displays the circle's diameter. Invoking **displayObject(r)** passes **r** to **g**. **g** is now an instance of **Rectangle**. The program displays the rectangle's width and height.

The isinstance Function

```
from Circle import Circle
from Rectangle import Rectangle


def displayObject(g):
     print ("Area is : ", g.getArea())
     print ("Perimeter is : ", g.getPerimeter())
     if isinstance(g, Circle):
          print ("Diameter is :"+ str (g.getDiamater()))
     elif isinstance(g, Rectangle):
          print ("Width is :"+str (g.getWidth()))
          print ("Heigth is :"+ str (g.getHeight()))

c = Circle (2)
displayObject(c)
r = Rectangle (3,2)
displayObject(r)
```

```
Area is :  12.566370614359172
Perimeter is :  12.566370614359172
Diameter is :4
Area is :  6
Perimeter is :  10
Width is :3
Heigth is :2
```

Exercise

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

- What is the output of this program below?

```
class Person:
    def getInfo(self):
        return "Person"

    def printPerson(self):
        print(self.getInfo())

class Student(Person):
    def getInfo(self):
        return "Student"

Person().printPerson()
Student().printPerson()
```

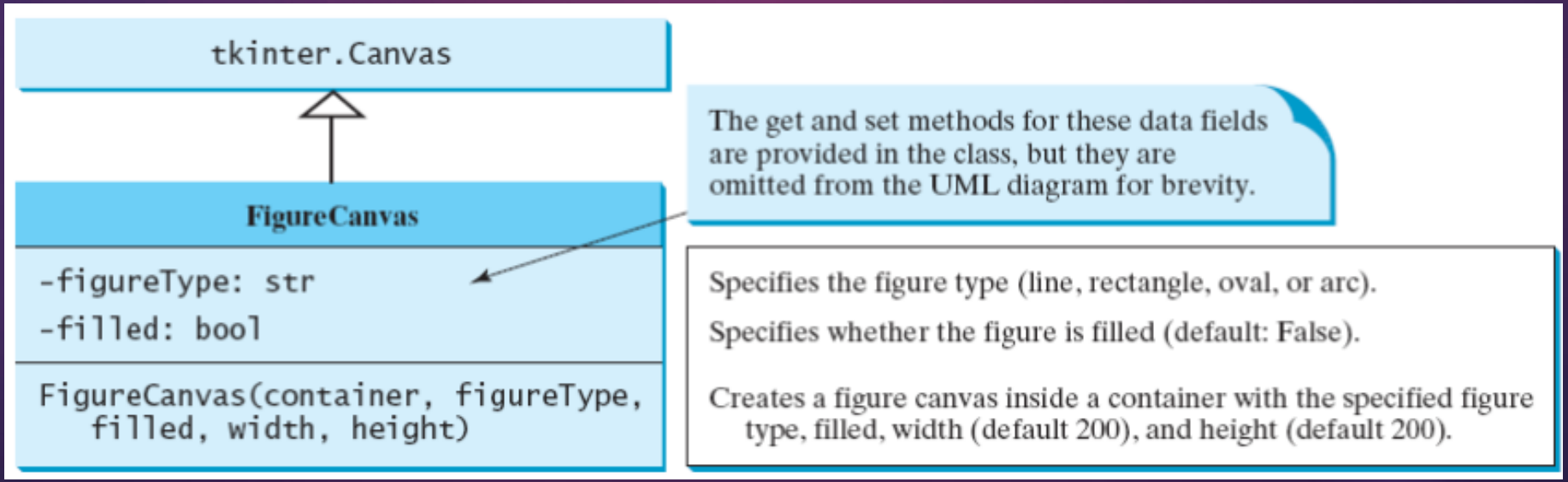
```
class Person:
    def __getInfo(self):
        return "Person"

    def printPerson(self):
        print(self.__getInfo())

class Student(Person):
    def __getInfo(self):
        return "Student"

Person().printPerson()
Student().printPerson()
```

Example FigureCanvas Class

- This case study develops the **FigureCanvas** class for displaying various figures.
- The **FigureCanvas** class enables the user to set the figure type, specify whether the figure is filled, and display the figure on a canvas.
- The UML diagram for the class, which can display lines, rectangles, ovals, and arcs, is shown in the next slide.
- The **figureType** property decides which figure to display. If the **filled** property is **True**, the rectangle, oval, or arc is filled with a color.

Example: FigureCanvas Class

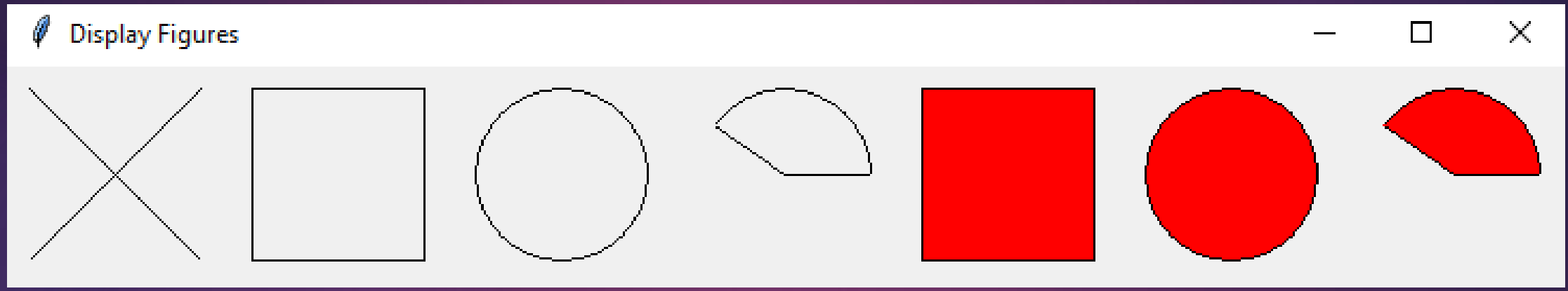
- The UML diagram serves as the contract for the **FigureCanvas** class.
- The user can use the class without knowing how the class is implemented
- The program in the next slide uses the class to display seven figures in a panel.



Example: FigureCanvas Class

- The **FigureCanvas** class extends the **Canvas** widget. Thus, a **FigureCanvas** is a canvas, and you can use **FigureCanvas** just like a canvas.
- You can construct a **FigureCanvas** by specifying the container, figure type, whether the figure is filled, and the canvas width and height.
- The **FigureCanvas** class's initializer invokes the **Canvas** initializer, sets the data field's **figureType** and **filled** properties, and invokes the **drawFigure** method to draw a figure.
- The **drawFigure** method draws a figure based on the **figureType** and **filled** properties.
- The methods **line**, **rectangle**, **oval**, and **arc** draw lines, rectangles, ovals, and arcs.

FigureCanvas



Example: Figures Canvas Class


```
from tkinter import * # Import tkinter
class FigureCanvas(Canvas):
    def __init__(self, container, figureType, filled = False,   width = 100,
height = 100):
        super().__init__(container,width = width, height = height)
        self.__figureType = figureType
        self.__filled = filled
        self.drawFigure()
    def getFigureType(self):
        return self.__figureType
    def getFilled(self):
        return self.__filled
    def setFigureType(self, figureType):
        self.__figureType = figureType
        self.drawFigure()
    def setFilled(self, filled):
        self.__filled = filled
        self.drawFigure()
```

Example: Figures Canvas Class

```
    def drawFigure(self):
        if self.__figureType == "line":
            self.line()
        elif self.__figureType == "rectangle":
            self.rectangle()
        elif self.__figureType == "oval":
            self.oval()
        elif self.__figureType == "arc":
            self.arc()
    def line(self):
        width = int(self["width"])
        height = int(self["height"])
        self.create_line(10, 10, width - 10, height - 10)
        self.create_line(width - 10, 10, 10, height - 10)
    def rectangle(self):
        width = int(self["width"])
        height = int(self["height"])
        if self.__filled:
            self.create_rectangle(10, 10, width - 10, height - 10,fill = "red")
        else:
            self.create_rectangle(10, 10, width - 10, height - 10)
```

Example: Figures Canvas Class

```
    def oval(self):
        width = int(self["width"])
        height = int(self["height"])
        if self.__filled:
            self.create_oval(10, 10, width - 10, height - 10,  fill = "red")
        else:
            self.create_oval(10, 10, width - 10, height - 10)
    def arc(self):
        width = int(self["width"])
        height = int(self["height"])
        if self.__filled:
            self.create_arc(10, 10, width - 10, height - 10, start = 0, extent = 145,
fill = "red")
        else:
            self.create_arc(10, 10, width - 10, height - 10, start = 0, extent = 145)
```

Example: Display Figures Class

```
#DisplayFigures.py
from tkinter import *
from FigureCanvas import FigureCanvas
class DisplayFigures:
    def __init__(self):
        window = Tk() # Create a window
        window.title("Display Figures") # Set title
        figure1 = FigureCanvas(window,"line",width=100,height=100)
        figure1.grid(row = 1, column = 1)
        figure2 = FigureCanvas(window, "rectangle", False, 100, 100)
        figure2.grid(row = 1, column = 2)
        figure3 = FigureCanvas(window, "oval", False, 100, 100)
        figure3.grid(row = 1, column = 3)
        figure4 = FigureCanvas(window, "arc", False, 100, 100)
        figure4.grid(row = 1, column = 4)
        figure5 = FigureCanvas(window, "rectangle", True, 100, 100)
        figure5.grid(row = 1, column = 5)
        figure6 = FigureCanvas(window, "oval", True, 100, 100)
        figure6.grid(row = 1, column = 6)
        figure7 = FigureCanvas(window, "arc", True, 100, 100)
        figure7.grid(row = 1, column = 7)
        window.mainloop() # Create an event loop
DisplayFigures() # Create GUI
```