



Taller 10

Fecha: Junio de 2021

Profesor: Fray León Osorio Rivera

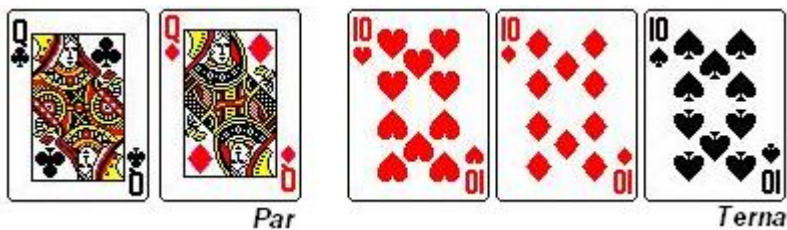
**Competencia a evaluar:** Aplicar los conceptos básicos de la orientación a objetos en el desarrollo de una aplicación con interfaz gráfica de usuario.

NOTAS:

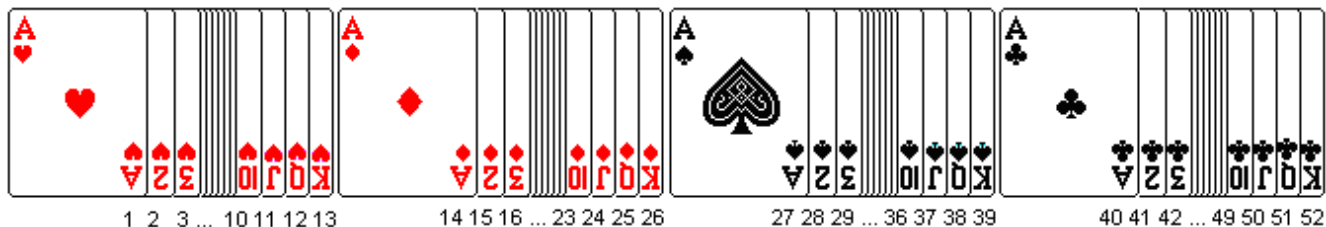
- Este taller se debe hacer como preparación para evaluaciones. En ningún caso representará una calificación.
- Los primeros ejercicios se entregan resueltos como ejemplo para el desarrollo de los demás

Elaborar el diagrama de clases básico (sin las clases correspondientes a la interface de usuario según el lenguaje de implementación) y la respectiva aplicación en un lenguaje orientado a objetos para los siguientes enunciados:

1. En un juego basado en la baraja inglesa, denominado “Apuntado”, se reparten a cada jugador 10 cartas. Cada jugador debe identificar las figuras que hay entre las cartas, es decir los agrupamientos por el nombre de la carta. Por ejemplo, si entre las 10 cartas hay 2 *Queen* (Q) independiente de la pinta, hay 1 par de Q. Si hay 3 *Diez* (10) con diferentes o iguales pintas, hay una terna de 10:



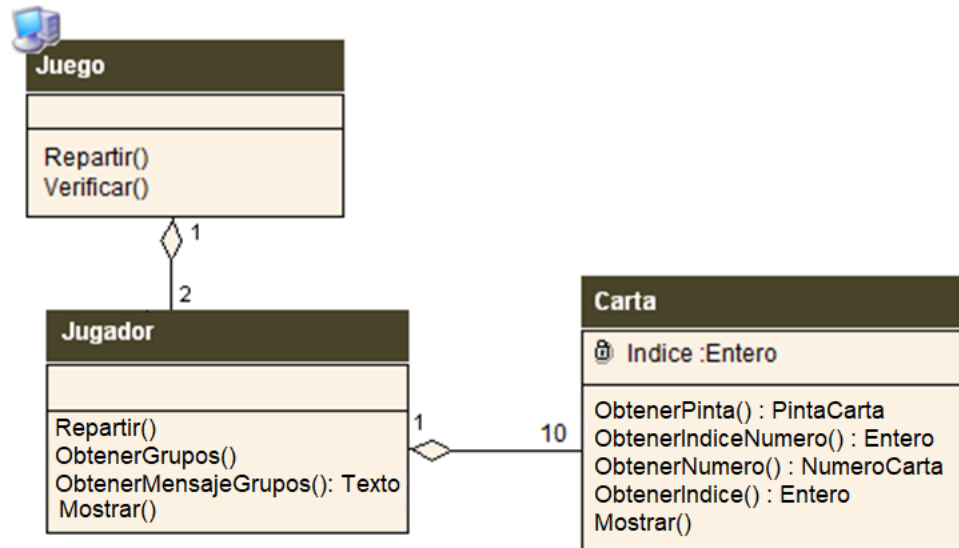
Para quien no conozca la baraja inglesa, esta se compone de un juego de 52 cartas que combina 13 nombres de cartas con 4 tipos de pinta:





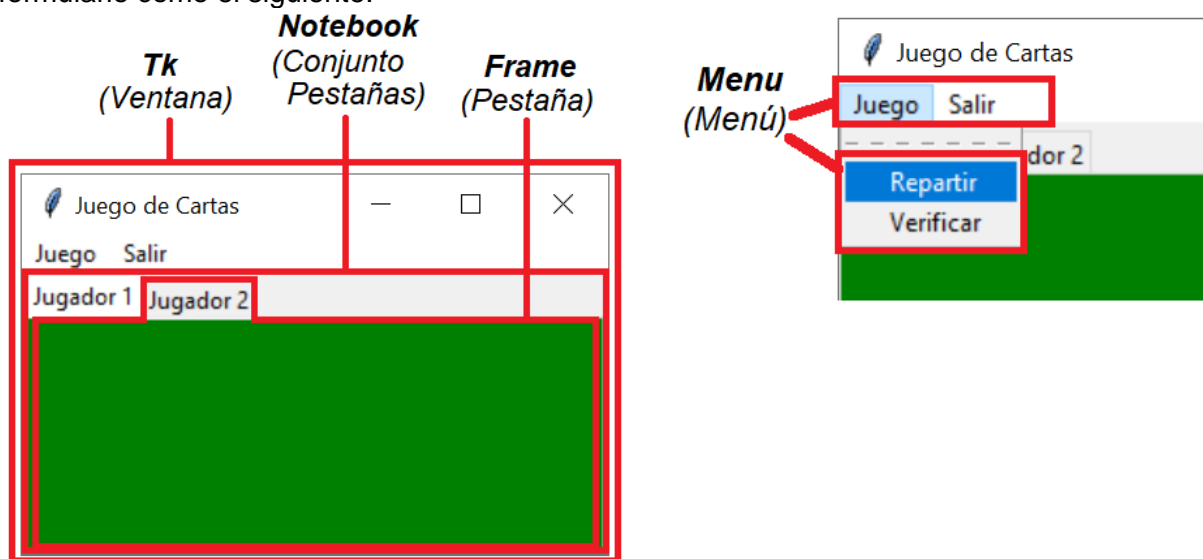
R/

- El modelado bajo el paradigma Orientado a Objetos se ilustra en el siguiente diagrama de clases:



### Programa

Para la implementación del aplicativo en *Python* se debe comenzar con el diseño de un formulario como el siguiente:



Este formulario corresponde a la clase *Juego* del anterior diagrama de clases. La siguiente tabla relaciona los objetos a añadir con las propiedades cuyos valores deben ser cambiados:

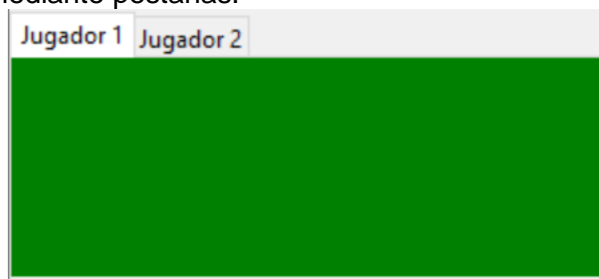
Tipo Control	Nombre	Otras Propiedades
<b>Tk</b>	<i>ventana</i>	title = "Juego de Cartas"
<b>Menu</b>	<i>mnuP</i>	label = "Juego", menú=mnuJ



		label = "Salir", command=salir
	<i>mnuJ</i>	label = "Repartir", command=repartir label = "Verificar", command=verificar
<b>Notebook</b>	<i>nbJ</i>	
<b>Frame</b>	<i>f1</i>	text = "Jugador 1"
	<i>f2</i>	text = "Jugador 2"

En este diseño de formulario aparecen unos nuevos tipos de objetos que se comportan como contenedores de otros objetos:

- **Notebook:** Corresponde a un agrupador de contenedores de objetos cuyos contenidos se pueden seleccionar mediante pestañas:



Cada contenedor asociado a una pestaña es un objeto **Frame**

- **Frame:** Corresponde a un contenedor de objetos. Predeterminadamente, toda ventana (objeto *Tk*) incluye uno:



De acuerdo con el modelo de clases planteado, se debe editar la clase *Carta* la cual tendrá la funcionalidad asociada a cada carta. Esta se compone de:

- La propiedad *indice* que identificará la carta de acuerdo con la siguiente convención:
  - Valores entre 1 y 13 para las cartas con pinta
  - Valores entre 14 y 26 para las cartas con pinta
  - Valores entre 27 y 39 para las cartas con pinta
  - Valores entre 40 y 52 para las cartas con pinta

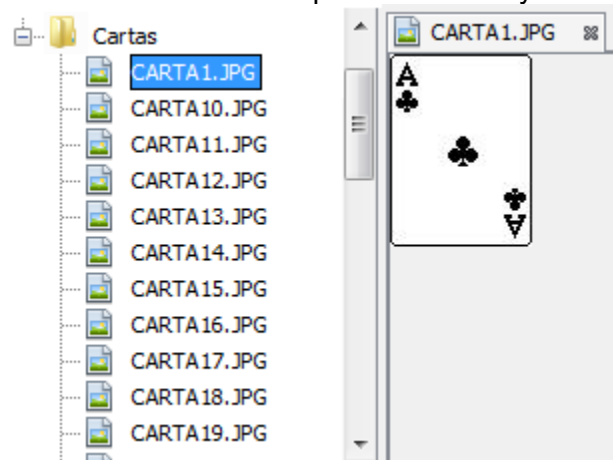
Es de observar que este valor debe coincidir con un conjunto de imágenes correspondientes a las 52 cartas en el siguiente orden:

Pinta	Nombre Carta												
	A	2	3	4	5	6	7	8	9	10	J	Q	K
	1	2	3	4	5	6	7	8	9	10	11	12	13



♠	14	15	16	17	18	19	20	21	22	23	24	25	26
♥	27	28	29	30	31	32	33	34	35	36	37	38	39
♦	40	41	42	43	44	45	46	47	48	49	50	51	52

Para lo cual se incluirá en el proyecto una carpeta denominada *Cartas* la cual incluirá las 52 imágenes cuyo nombre se formará con la palabra “*Carta*” y el respectivo índice:



Para efectos de controlar las listas de pintas, de los nombres de las cartas y los tipos de grupos, se utilizará el tipo **Enum**. Esto posibilita que, en lugar de utilizar un texto o número por cada pinta o nombre de carta, se utilice un valor constante que puede ser manipulado por su posición, además de otras posibilidades.

La siguiente es la declaración de los enumerados requeridos:

```
from enum import Enum

class NumeroCarta(Enum):
    NINGUNO = 0
    AS = 1
    DOS = 2
    TRES = 3
    CUATRO = 4
    CINCO = 5
    SEIS = 6
    SIETE = 7
    OCHO = 8
    NUEVE = 9
    DIEZ = 10
    JACK = 11
    QUEEN = 12
    KING = 13


class PintaCarta(Enum):
    NINGUNO = 0
    TREBOL = 1
    PICA = 2
    CORAZON = 3
    DIAMANTE = 4
```



```
class GrupoCarta(Enum):
    NINGUNA = 0
    NON = 1
    PAR = 2
    TERNA = 3
    CUARTA = 4
    QUINTA = 5
    SEXTA = 6
    SEPTIMA = 7
    OCTAVA = 8
    NOVENA = 9
    DECIMA = 10
```

Obsérvese que un enumerado se declara como una clase que hereda de la clase **Enum**.

Algún valor de las anteriores listas será el que devuelvan algunos de los métodos de la clase, como se podrá verificar a continuación:

Carta
 <b>Indice :Entero</b>
<b>ObtenerPinta() : PintaCarta</b> <b>ObtenerIndiceNumero() : Entero</b> <b>ObtenerNumero() : NumeroCarta</b> <b>ObtenerIndice() : Entero</b> <b>Mostrar()</b>

- El método *Constructor* (utilizado en la instrucción **\_\_init\_\_**), que como se sabe, es invocado cuando se crea un objeto de la clase, generará aleatoriamente el índice de la carta.
- El método *mostrar()* permite ver la imagen de la carta que corresponda al *indice*.
- El método *obtenerNumero()* devuelve un valor del enumerado *NumeroCarta* con base en el *indice*.

- El método *obtenerIndiceNumero()* devuelve el índice del respectivo *NumeroCarta* con base en el *indice*
- El método *obtenerPinta()* devuelve un valor del enumerado *PintaCarta* con base en el *indice*
- El método *obtenerIndice()* devuelve el *indice* que esta encapsulado

El código respectivo sería el siguiente:

```
#Importar la libreria para GUI
from tkinter import *

#Importar generador de numeros aleatorios
import random

#Importar los Enumerados
from Enumerados import *

class Carta:
    #Metodo constructor
    def __init__(varClase):
        #Generar aleatoriamente el indice de la carta
```



```

varClase.indice=random.randrange(1, 53)

def obtenerPinta(varClase):
    if varClase.indice <= 13:
        return PintaCarta.TREBOL
    elif indice <= 26:
        return PintaCarta.PICA;
    elif indice <= 39:
        return PintaCarta.CORAZON;
    else:
        return PintaCarta.DIAMANTE;

def obtenerNumero(varClase):
    n = varClase.indice % 13
    if n==0:
        n=13;
    return NumeroCarta(n)

def mostrar(varClase, frm, x, y):
    lblCarta=Label(frm)
    #cargar la imagen
    imgCarta=PhotoImage(file = "../Carta"+str(varClase.indice)+".gif")
    #Mostrar la imagen
    lblCarta.config(image=imgCarta)
    lblCarta.image=imgCarta
    lblCarta.place(x=x, y=y)

def obtenerIndice(varClase):
    return varClase.indice

```

En este código se pueden apreciar los siguientes detalles de implementación:

- El método *mostrarCarta()* recibe como parámetros de entrada: Las coordenadas donde se ubicará el objeto que mostrará la imagen de la carta (un *Label*) el cual a su vez se incluirá en un objeto *Frame*.
- La divisibilidad por 13 es un factor clave para obtener el numero de la carta y su respectivo valor a partir del *Indice*. Esto se debe a que el total de cartas por pinta es 13.

Siguiendo con el modelo de clases planteado, se debe editar la clase *Jugador* la cual tendrá la funcionalidad asociada a cada jugador. Se debe tener en cuenta que esta clase es un *agregado* de objetos de la clase *Carta*. Esta se compone de:

Jugador
Repartir() ObtenerGrupos() ObtenerMensajeGrupos(): Texto

- El método *repartir()* que permite crear las instancias de las 10 cartas, simulando la repartición de cartas. Es preciso tener en cuenta que en cada objeto *Carta* generado, el valor del *indice* que la identifica es aleatorio (Ver el respectivo método constructor).
- El método *mostrarCarta()* permite ver las imágenes de las cartas que tiene el jugador.



- El método *obtenerMensajeGrupos()* devuelve un texto que indica los grupos encontradas, es decir, si hubo pares, ternas, cuartas, etc. y de que nombres de cartas

El código respectivo sería el siguiente:

```
#importar la clase CARTA
from Carta import Carta

#Importar los Enumerados
from Enumerados import *

class Jugador :

    #Metodo constructor
    def __init__(varClase):
        varClase.cartas=[]
        varClase.totalCartas = 10
        varClase.Grupos=[]
        varClase.numeroGrupos=[]

    #Metodo para generar las cartas
    def repartir(varClase):
        varClase.cartas = []
        for i in range(0, varClase.totalCartas+1):
            varClase.cartas.append(Carta())

    #Metodo para mostrar todas las cartas del jugador
    def mostrarCartas(varClase, frm) :
        #limpiar el panel
        for w in frm.wininfo_children():
            w.destroy()
        x = 5
        for i in range(0, len(varClase.cartas)):
            varClase.cartas[i].mostrar(frm, x, 5)
            x += 45

    #Metodo para encontrar las Grupos que hay en las cartas
    def obtenerGrupos(varClase) :
        varClase.Grupos = []
        varClase.numeroGrupos = []

        contadores = []
        for i in range(0, 13):
            contadores.append(0)

        for i in range(0, len(varClase.cartas)):
            pc=varClase.cartas[i].obtenerNumero() - 1
            contadores[pc] += 1;

        #Contar cuantas Grupos hay
        cf = 0
        for i in range(0, 13):
```



```

        if contadores[i] > 1:
            cf+=1

    #Hubo Grupos?
    if (cf > 0) :
        #Instanciar las Grupos
        cf = 0;
        for i in range(0, 13):
            if (contadores[i] > 1) :
                pf=contadores[i]

                varClase.Grupos.append(GrupoCarta(pf))
                varClase.numeroGrupos.append(NumeroCarta(i + 1))

def obtenerMensajeGrupos(varClase):
    mensaje="No hay Grupos"
    varClase.obtenerGrupos();
    if len(varClase.Grupos)>0:
        mensaje="Las Grupos del jugador son:\n"
        for i in range(0, len(varClase.Grupos)):
            mensaje+=varClase.Grupos[i].name + " de
"+varClase.numeroGrupos[i].name +"\n"
    return mensaje

```

En este código se pueden apreciar los siguientes detalles de implementación:

- La relación de agregación con la clase *Carta* se representa mediante un vector de estos objetos, lo cual requiere de la respectiva instrucción de declaración
- El método *mostrarCarta()* recibe como parámetros un objeto *Frame* que contendrá las imágenes de las cartas.
- La estrategia para encontrar las figuras en el método *obtenerGrupos()* utiliza un vector de enteros en el que se cuenta cuanto aparece cada nombre de carta (En total serían 13 contadores). Luego se verifica que contadores son iguales o superiores a 2 para determinar el grupo y concatenar los textos que así lo indiquen.

Ahora bien, para continuar con la codificación, se deben programar los métodos *Repartir()* y *Verificar()* de la clase *Juego* y que corresponderán a los eventos de los botones de comando agregados al formulario en la barra de herramientas.

El código respectivo sería el siguiente:

```

j1=Jugador()
j2=Jugador()

def repartir():
    j1.repartir()
    j1.mostrarCartas(f1)
    j2.repartir()
    j2.mostrarCartas(f2)

def verificar():
    if nbJ.index(nbJ.select())==0:

```





```
f=j1.obtenerMensajeGrupos()  
else:  
    f=j2.obtenerMensajeGrupos()  
messagebox.showinfo("Verificacion", f)
```

En este código se invocan los métodos *repartir()* y *mostrar()* de cada uno de los objetos de la clase *Jugador* cuyas instancias se declaran globalmente antes de los métodos:

En el código de verificación se pregunta por la pestaña del jugador visible para poder encontrar las figuras de las cartas que son visibles.

La ejecución de la aplicación luciría así cuando se reparten las cartas:



Cuando se verifica las figuras del jugador seleccionado, podría lucir así:





2. Basado en el ejercicio anterior, incluir la funcionalidad que permita:
- Obtener las figuras en escalera de la misma pinta, es decir, secuencias de cartas que tengan la misma pinta. Por ejemplo: 10, J, Q y K de Pica conforman una cuarta de pica.



- Calcular el puntaje del jugador con base en el valor de las cartas que no conforman figuras, teniendo en cuenta que las cartas ("Ace", "Jack", "Queen", y "King" valen 10 y el resto, el respectivo número del nombre.