2.5. La clase Pack

Todos los widgets derivan de la clase Widget, por lo que tienen definidos los métodos de la clase Pack, entre ellos pack(). Al crear un widget se ha de indicar quién es su padre, para colocarlo dentro del padre podemos usar este método (*).

Para entender cómo coloca este método los widgets dentro de su padre hemos de pensar que el padre es un rectángulo elástico, en principio vacío (de 0 x 0 píxeles), que va agrandándose conforme le añadimos contenido.

```
Label(v, bg='red').pack()
Label(v, bg='green', text='Etiqueta').pack()
Label(v, bg='red', text='Muchas\nlíneas\n...\n\n\n').pack()
```

tkinter decide el tamaño de cada etiqueta en función de su contenido, las coloca centradas horizontalmente en la ventana principal (su padre) conforme se lo decimos: coloca una etiqueta vacía en la parte superior del padre, luego otra etiqueta en la parte superior del padre (que queda debajo de la que ya había colocado) y finalmente una etiqueta multilínea en la parte superior del padre (que queda debajo de las que ya se han colocado en esa parte). Este párrafo se entiende mejor cuando se han hecho muchas pruebas, yo lo intento explicar en clase usando matices que no sé explicar aquí.

Al colocar el widget podemos configurar diferentes opciones:

- **after y before** nos permiten cambiar el orden de colocación respecto a otro widget ya colocado en el mismo padre.
- **side** indica el lado desde el que vamos apilando los widgets en su padre. Puede tomar los valores top (por defecto), bottom, left o right. Si se colocan varios widgets en el mismo padre desde diferentes lados, tkinter hace bien su trabajo y los muestra todos bien, pero si se redimensiona la ventana la experiencia de usuario es desastrosa. Se recomienda usar frames para colocar siempre los widgets de un padre desde el mismo lado.
- **padx, pady** son los espacios horizontal y vertical que tiene exteriormente el widget al colocarlo. Por defecto ambos valen 0. Ver también ipadx e ipady.
- **fill** indica si el widget debe aumentar su tamaño para rellenar el espacio de su padre. Puede tomar los valores none, x, y y both.

(*) Más adelante estudiaremos otros métodos con el mismo fin.

`expand` es un booleano que indica si el widget debe expandirse o no cuando el usuario redimensiona a su padre.

`anchor` ancla el widget a una parte del padre: `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, `nw` o `center`. Tiene efecto cuando el usuario redimensiona a su padre.

Pack define funciones que pueden usar todos los widgets, tanto padres como hijos. Todos tienen el prefijo `pack_`. Por ejemplo, si queremos desempaquetar la etiqueta `l1` usamos su método `pack_forget()`. Si queremos volver a colocarla podemos usar de nuevo su método `pack()`, y se colocará después del último widget colocado en su padre (después de `l2` en el ejemplo). Podemos lograr el mismo resultado usando sólo una función, `pack_configure()`, que permite configurar el empaquetado del widget y al que podemos indicarle que queremos que esté después de `l2`, simplemente escribiendo

```
l1 = Label(v, text='1')
l1.pack()
l2 = Label(v, text='2')
l2.pack()
```

```
l1.pack_configure(after=l2)
```

La función `pack_slaves()` devuelve una lista con los widgets que le hemos colocado con `pack()`. Este código

```
for w in v.pack_slaves():
    print(w['text'])
```

escribiría `1 2` si lo ejecutamos antes que la función anterior. Si lo hacemos después nos devolvería `2 1`. Estas funciones no se usan en la mayoría de aplicaciones, pero hay casos en que, por ejemplo, queremos mostrar u ocultar un widget en función de lo que decida el usuario (con una casilla de verificación, por ejemplo).