Copilot

Sure, I can help with that. Here's how you can modify the code to resize and reposition the circles when the window is resized:

```
import tkinter as tk

class Application(tk.Frame):
    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.pack(fill="both", expand=True)
        self.create_widgets()

    def create_widgets(self):
        self.button_frame = tk.Frame(self)
        self.button_frame.pack(side="top", fill="x")

        self.add_button = tk.Button(self.button_frame)
        self.add_button["text"] = "Add"
        self.add_button["command"] = self.enable_add
        self.add_button.pack(side="left")

        self.delete_button = tk.Button(self.button_frame)
        self.delete_button["text"] = "Delete"
        self.delete_button["command"] = self.enable_delete
        self.delete_button.pack(side="right")

        self.canvas = tk.Canvas(self, bg='white')
        self.canvas.pack(fill="both", expand=True)
        self.canvas.bind("<Button-1>", self.draw_or_delete_circle)
        self.canvas.bind("<Configure>", self.resize_circles)

        self.adding = False
        self.deleting = False
        self.circles = []

    def enable_add(self):
        self.adding = True
        self.deleting = False
        self.add_button.config(relief="sunken")
        self.delete_button.config(relief="raised")

    def enable_delete(self):
        self.adding = False
        self.deleting = True
        self.add_button.config(relief="raised")
        self.delete_button.config(relief="sunken")

    def draw_or_delete_circle(self, event):
        x, y = event.x, event.y
        r = 10  # radius
        overlapping = self.canvas.find_overlapping(x-r, y-r, x+r, y+r)
        if self.adding:
            id = self.canvas.create_oval(x-r, y-r, x+r, y+r, fill="blue")
            self.circles.append((id, x, y, r))
        elif self.deleting and overlapping:
            for circle in self.circles:
                if circle[0] in overlapping:
                    self.canvas.delete(circle[0])
                    self.circles.remove(circle)

    def resize_circles(self, event):
        for circle in self.circles:
            id, x, y, r = circle
            self.canvas.coords(id, x-r, y-r, x+r, y+r)

root = tk.Tk()
app = Application(master=root)
app.mainloop()
```

In this updated code, I've added a `circles` list to keep track of the circles that have been added to the canvas. Each circle is represented as a tuple of the form `(id, x, y, r)`, where `id` is the ID of the circle returned by `create_oval`, `x` and `y` are the coordinates of the center of the circle, and `r` is the radius of the circle. When a circle is added, it is appended to the `circles` list, and when a circle is deleted, it is removed from the `circles` list. The `resize_circles` method is bound to the `<Configure>` event of the canvas, which is triggered when the canvas is resized. This method iterates over the `circles` list and updates the coordinates of each circle to maintain its relative position and size on the canvas. Please note that this code assumes that the aspect ratio of the window remains constant when it is resized. If the window can be resized in a way that changes its aspect ratio (for example, by dragging one of its corners), the circles will become ellipses. If you want to maintain the circles as circles even when the aspect ratio of the window changes, you would need to implement additional logic. I hope this helps! Let me know if you have any other questions.