slip29

```
import java.util.Scanner;
class NotEligible extends Exception {
   public NotEligible(String message) {
      super(message);
   }
}

class s29q1 {
  public static void checkEligibility(int age) throws NotEligible {
      if (age < 18) {

         throw new NotEligible("Candidate is not eligible for voting. Age is below 18.");
      } else {
         System.out.println("Candidate is eligible for voting.");
      }
   }

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      try {

         System.out.print("Enter the candidate's age: ");
         int age = Integer.parseInt(scanner.nextLine());
```

```
            if (age <= 0) {
                throw new IllegalArgumentException("Age must be a positive number.");
            }

             checkEligibility(age);
        } catch (NotEligible e) {
                System.out.println("Exception: " + e.getMessage());
        } catch (NumberFormatException e) {
                 System.out.println("Exception: Invalid input. Please enter a valid integer for age.");
        } catch (IllegalArgumentException e) {
                 System.out.println("Exception: " + e.getMessage());
        } finally {
                scanner.close();
        }
    }
}
```

```
import java.applet.Applet;
import java.awt.*;
import java.util.Random;

public class s29q2 extends Applet implements Runnable {
```

```
private int x = 50, y = 50; // Ball position
private int radius = 20;    // Ball radius
private int dx = 2, dy = 2; // Ball movement direction
private Color ballColor;    // Ball color
private Thread animationThread;
private Random random;

// Initialize the applet
public void init() {
    setSize(400, 300); // Set applet size
    ballColor = Color.RED; // Initial color
    random = new Random(); // Random object for color generation
    animationThread = new Thread(this);
    animationThread.start(); // Start the animation thread
}

// Paint the ball
public void paint(Graphics g) {
    g.setColor(ballColor);
    g.fillOval(x, y, radius * 2, radius * 2); // Draw the ball
}

// Run the animation
public void run() {
    while (true) {
```

```
    // Move the ball
    x += dx;
    y += dy;

    // Check for bounce on the left or right boundaries
    if (x < 0 || x > getWidth() - radius * 2) {
        dx = -dx; // Reverse horizontal direction
        changeBallColor(); // Change color on bounce
    }

    // Check for bounce on the top or bottom boundaries
    if (y < 0 || y > getHeight() - radius * 2) {
        dy = -dy; // Reverse vertical direction
        changeBallColor(); // Change color on bounce
    }

    // Repaint the applet
    repaint();

    // Delay to control the speed of the ball
    try {
        Thread.sleep(10);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

```
  }

  // Change the ball's color to a random color
  private void changeBallColor() {
    ballColor = new Color(random.nextInt(256), random.nextInt(256),
random.nextInt(256));
  }
}
<html>
  <body>
    <applet code="s29q2.class" width="400" height="300">
    </applet>
  </body>
</html>
```

```
import tkinter as tk
from tkinter import messagebox
import math
class SphereVolumeCalculator:
  def __init__(self, master):
    self.master = master
    self.master.title("Sphere Volume Calculator")

    self.label = tk.Label(master, text="Enter the radius of the sphere:")
```

```
        self.label.pack(pady=10)

        self.entry = tk.Entry(master)
        self.entry.pack(pady=5)

        self.calculate_button = tk.Button(master, text="Calculate Volume",
command=self.calculate_volume)
        self.calculate_button.pack(pady=20)

        self.result_label = tk.Label(master, text="")
        self.result_label.pack(pady=10)

    def calculate_volume(self):
        radius_str = self.entry.get()

        try:
            radius = float(radius_str)
            if radius < 0:
                raise ValueError("Please enter a non-negative radius.")
        except ValueError as e:
            messagebox.showerror("Invalid input", str(e))
            return
            volume = (4/3) * math.pi * (radius ** 3)

        self.result_label.config(text=f"Volume of the sphere: {volume:.2f}")

root = tk.Tk()
app = SphereVolumeCalculator(root)
```

```
root.mainloop()
```

```
def sort_dictionary(d):
    sorted_by_keys_asc = dict(sorted(d.items()))
    sorted_by_keys_desc = dict(sorted(d.items(), reverse=True))

    sorted_by_values_asc = dict(sorted(d.items(), key=lambda item: item[1]))
    sorted_by_values_desc = dict(sorted(d.items(), key=lambda item: item[1],
reverse=True))

    return (sorted_by_keys_asc, sorted_by_keys_desc, sorted_by_values_asc,
sorted_by_values_desc)

def main():
    sample_dict = {
        'banana': 3,
        'apple': 4,
        'orange': 2,
        'kiwi': 5,
        'mango': 1
    }

    sorted_results = sort_dictionary(sample_dict)
```

```
    print("Original Dictionary:", sample_dict)
    print("\nSorted by Keys (Ascending):", sorted_results[0])
    print("Sorted by Keys (Descending):", sorted_results[1])
    print("Sorted by Values (Ascending):", sorted_results[2])
    print("Sorted by Values (Descending):", sorted_results[3])

main()
```