Question 1:

This code is designed for representing and manipulating 2D points using the Point2D class. The Point2D class encapsulates an x and y coordinate and includes constructors for default initialization, explicit coordinate setting, and copy creation. It provides methods like input() to allow user modification, isOrigin() to check if the point is at (0,0), and a toString() method for easy printing. Crucially, it offers both an instance method and a static method for calculating the Euclidean distance between two points. The TestingPoint2D class serves as a driver to demonstrate these functionalities, creating, modifying, and calculating distances for several Point2D objects.

Question 2:

This code introduces a Triangle class built upon the previously defined Point2D class from question1 to model a triangle in a 2D plane. The Triangle class stores its three vertices as Point2D objects and has a constructor to initialize them. Its core functionality lies in two methods: perimeter(), which calculates the total length of the sides by summing the distances between the vertices, and area(), which computes the area using Heron's formula. The separate TriangleTesting class serves as the driver, demonstrating how to instantiate a Triangle with specific Point2D coordinates and then prints both its calculated perimeter and area to the console. This showcases code reuse and object composition by using the Point2D class to build a more complex geometric shape.

Question 3:

This code models a basic 2D particle collision and multiplication simulation. The Direction enum defines the eight possible movement vectors for a particle. Each Particle randomly moves one step in one of these directions during a simulation step, bounded by the Box dimensions, and can check for collision with another particle using a fixed $\mathbf{1.5}$ unit radius. The Box class, which uses the Singleton pattern, manages the collection of particles and runs the simulation logic in its step() method. In each step, every particle moves, and if any two particles collide, a new random particle is generated and added to the box. The Simulation driver initializes the box and iteratively calls the step() and visualize() methods, displaying the particle locations until a maximum of 20 particles is reached.