```java
import java.util.ArrayList;
import java.util.InvalidUserDefinedArrayException;
import java.util.Scanner;

/**
 * Interface defining coffee operations.
 */
interface CoffeeOperations {
    void Brew();
    String describe();
}


/**
 * Main CLASS Program for managing Coffee objects.
 * Supports adding, viewing, updating, and deleting coffees.
 */
public final class CoffeeMaker { // Recommended: Make this final if not a base class
    private static ArrayList<Coffee> coffeeList = new ArrayList<>();
    private static Scanner scanner = new Scanner(System.in);
    private static int coffeeCount = 0; // Static variable to count coffee objects

    public static void main(String[] args) {
        int choice;
        do {
            System.out.println("\n=== Coffee Menu ===");
            System.out.println("1. Add Coffee");
            System.out.println("2. View Coffees");
            System.out.println("3. Update Coffee");
            System.out.println("4. Delete Coffee");
            System.out.println("5. Exit");
            System.out.print("Choose an option: ");
            choice = readIntInput();

            switch (choice) {
                case 1 -> addCoffee();
                case 2 -> viewCoffees();
                case 3 -> updateCoffee();
                case 4 -> deleteCoffee();
                case 5 -> System.out.println("Exiting program...");
                default -> System.out.println("Invalid choice. Please try again.");
            }
        } while (choice != 5);
    }


    /**
     * Reads an integer input from the user with validation.
     * @return valid integer input
     */
    private static int readIntInput() {
        while (true) {
            try {
                int input = scanner.nextInt();
                scanner.nextLine();
                return input;
            } catch (InputMismatchException e) {
                System.out.print("Invalid input. Please enter a valid integer: ");
                scanner.nextLine(); // Clear invalid input
            }
        }
    }


    /**
     * Reads a double input from the user with validation.
     * @return valid double input
     */
    private static double readDoubleInput() {
        while (true) {
            try {
                double input = scanner.nextDouble();
                scanner.nextLine();
                return input;
            } catch (InputMismatchException e) {
                System.out.print("Invalid input. Please enter a valid number: ");
                scanner.nextLine(); // Clear invalid input
            }
        }
    }


    /**
     * Adds a new coffee to the list.
     * Supports Espresso and Latte types.
     */
    private static final void addCoffee() { // Recommended: Make this final if not a base class
        System.out.println("Choose coffee type:");
        System.out.println("1. Espresso");
        System.out.println("2. Latte");
        System.out.print("Enter choice: ");
        int typeChoice = readIntInput();

        System.out.print("Enter coffee name: ");
        String name = scanner.nextLine();
        System.out.print("Enter coffee origin: ");
        String origin = scanner.nextLine();
        System.out.print("Enter coffee price: ");
        double price = readDoubleInput();

        Coffee newCoffee;
        if (typeChoice == 1) {
            System.out.print("Enter coffee shots (kg): ");
            int coffeeShots = readIntInput();
            newCoffee = new Espresso(name, origin, price, coffeeShots);
        } else if (typeChoice == 2) {
            System.out.print("Enter milk ratio (0.0 - 1.0): ");
            double milkRatio = readDoubleInput();
            if (milkRatio < 0.0 || milkRatio > 1.0) {
                System.out.println("Invalid milk ratio. Must be between 0.0 and 1.0.");
                return;
            }
            newCoffee = new Latte(name, origin, price, milkRatio);
        } else {
            System.out.println("Invalid selection.");
            return;
        }

        coffeeList.add(newCoffee);
        coffeeCount++; // Increment static coffee count
        System.out.println("Coffee added successfully! Total coffees: " + coffeeCount);
    }

    /**
     * Displays the list of coffees.
     */
    private static void viewCoffees() {
        if (coffeeList.isEmpty()) {
            System.out.println("No coffees available.");
            return;
        }
        System.out.println("List of Coffees:");
        for (int i = 0; i < coffeeList.size(); i++) {
            System.out.print(i + 1) + ". " + coffeeList.get(i).describe());
        }
    }


    /**
     * Updates an existing coffee's price and subclass-specific attributes.
     * Name and origin cannot be changed.
     */
    private static void updateCoffee() {
        if (coffeeList.isEmpty()) {
            System.out.println("No coffees available to update.");
            return;
        }
        viewCoffees();
        System.out.print("Enter the number of the coffee to update: ");
        int index = readIntInput() - 1;
        if (index < 0 || index >= coffeeList.size()) {
            System.out.println("Invalid coffee number.");
            return;
        }
        Coffee coffee = coffeeList.get(index);
        System.out.println("Updating: " + coffee.describe());

        System.out.print("Enter new name (press Enter to keep current): ");
        String newName = scanner.nextLine();
        System.out.print("Enter new origin (press Enter to keep current): ");
        String newOrigin = scanner.nextLine();
        System.out.print("Enter new price (-1 to keep current): ");
        double newPrice = readDoubleInput();

        if (!newName.isEmpty()) {
            System.out.println("Name cannot be changed.");
        }
        if (!newOrigin.isEmpty()) {
            System.out.println("Origin cannot be changed.");
        }
        if (newPrice >= 0) {
            coffee.setPrice(newPrice);
        }

        if (coffee instanceof Espresso espresso) {
            System.out.print("Enter new coffeeShots level (-1 to keep current): ");
            int newCoffeeShots = readIntInput();
            if (newCoffeeShots >= 0) {
                espresso.setCoffeeShotsLevel(newCoffeeShots);
            }
        } else if (coffee instanceof Latte latte) {
            System.out.print("Enter new milk ratio (-1 to keep current): ");
            double newMilkRatio = readDoubleInput();
            if (newMilkRatio >= 0) {
                if (newMilkRatio >= 0.0 && newMilkRatio <= 1.0) {
                    latte.setMilkRatio(newMilkRatio);
                } else {
                    System.out.println("Invalid milk ratio. Must be between 0.0 and 1.0.");
                }
            }
        }
        System.out.println("Coffee updated successfully!");
    }

    /**
     * Deletes a coffee from the list.
     */
    private static void deleteCoffee() {
        if (coffeeList.isEmpty()) {
            System.out.println("No coffees available to delete.");
            return;
        }
        viewCoffees();
        System.out.print("Enter the number of the coffee to delete: ");
        int index = readIntInput() - 1;
        if (index < 0 || index >= coffeeList.size()) {
            System.out.println("Invalid coffee number.");
            return;
        }
        coffeeList.remove(index);
        coffeeCount--; // Decrement static coffee count
        System.out.println("Coffee deleted successfully! Total coffees: " + coffeeCount);
    }
}

// Espresso subclass
class Espresso private Coffee implements CoffeeOperations {
    private int coffeeShots;

    public Espresso(String name, String origin, double price, int coffeeShots) {
        super(name, origin, price);
        this.coffeeShots = coffeeShots;
    }

    @Override
    public void Brew() {
        System.out.println("Brewing a strong Espresso...");
    }

    public int getCoffeeShots() { return coffeeShots; }
    public void setCoffeeShots(int coffeeShots) { this.coffeeShots = coffeeShots; }

    @Override
    public String describe() {
        return super.toString() + ", Espresso{coffeeShots=" + coffeeShots + "kg}";
    }
}

// Specialized Latte
class Latte extends Coffee implements CoffeeOperations {
    private double milkRatio;

    public Latte(String name, String origin, double price, double milkRatio) {
        super(name, origin, price);
        this.milkRatio = milkRatio;
    }

    @Override
    public void Brew() {
        System.out.println("Brewing a creamy Latte...");
    }

    public double getMilkRatio() { return milkRatio; }
    public void setMilkRatio(double milkRatio) { this.milkRatio = milkRatio; }

    @Override
    public String describe() {
        return super.toString() + ", Latte{milkRatio=" + (int)(milkRatio * 100) + "%";
    }
}

// Base Coffee class (added for completeness)
abstract class Coffee implements CoffeeOperations {
    private final String name;
    private final String origin;
    private double price;

    public Coffee(String name, String origin, double price) {
        this.name = name;
        this.origin = origin;
        this.price = price;
    }

    public String getName() { return name; }
    public String getOrigin() { return origin; }
    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }

    @Override
    public abstract void Brew();

    @Override
    public String describe() {
        return "Coffee{name=" + name + ", origin=" + origin + ", price=" + price + "}";
    }
}
```