Here's the Java implementation based on the provided prompt:

Java

```java
class User {
    String firstName, lastName, phoneNumber;

    // Constructors and getters/setters
    public User(String firstName, String lastName, String phoneNumber) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.phoneNumber = phoneNumber;

    }

    // Getters and setters
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;

    }

    // ... other getters and setters

}

class Fruit {
    String name, variety;
    double totalQuantityKg, pricePerKg;

    // Constructors and getters/setters
    public Fruit(String name, String variety, double totalQuantityKg, double pricePerKg) {
        this.name = name;
        this.variety = variety;
        this.totalQuantityKg = totalQuantityKg;
        this.pricePerKg
 = pricePerKg;
    }

    // Getters and setters
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
    // ... other getters and setters
}

class Seller extends User {
    String address;
    List<Fruit> fruits;

    // Constructors and getters/setters
    public Seller(String firstName, String lastName, String phoneNumber, String address, Lis
        super(firstName, lastName, phoneNumber);
        this.address = address;
        this.fruits = fruits;
    }

    // Getters and setters
    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public
 List<Fruit> getFruits() {
        return fruits;
    }

    public void setFruits(List<Fruit> fruits) {
        this.fruits = fruits;

    }
}

class Order {
    List<Fruit> fruits;
    double quantityInKg;
    Buyer buyer;
    Seller seller;

    // Constructors and getters/setters
}

class Buyer extends User {
    String address;
    int rewards;
    Order order;

    // Constructors and getters/setters
    public Buyer(String firstName, String lastName, String phoneNumber, String address) {
        super(firstName, lastName, phoneNumber);
        this.address = address;

        this.rewards = 0;
    }

    // Getters and setters
    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public
int getRewards() {
        return rewards;
    }

    public void setRewards(int rewards) {
        this.rewards = rewards;
    }

    public Order placeOrder(Seller seller) {
        // Display seller's fruit list
        System.out.println("Seller's Fruits:");
        for (Fruit fruit : seller.getFruits()) {
            System.out.println("- " + fruit.getName() + " (" + fruit.getVariety() + "): " +
        }

        // Take user input for fruit name and quantity
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter fruit name: ");
        String fruitName = scanner.nextLine();
        System.out.print("Enter quantity in kg: ");
        double quantity = scanner.nextDouble();

        // Find the fruit in the seller's list
        Fruit selectedFruit = null;
        for (Fruit fruit : seller.getFruits()) {
            if (fruit.getName().equals(fruitName)) {
                selectedFruit = fruit;
                break;
            }
        }

        // Check if the fruit exists and has sufficient quantity
        if (selectedFruit == null || selectedFruit.getTotalQuantityKg() < quantity) {
            throw new NoFruitsException("Fruit not found or insufficient quantity.");
        }

        // Create and return Order object
        Order order = new Order();
        order.setFruits(List.of(selectedFruit));
        order.setQuantityInKg(quantity);
        order.setBuyer(this);
        order.setSeller(seller);
        return order;
    }
}

interface RoutePlanner {
    String calculateRoute(String sourceAddress, String destinationAddress);
}

class Delivery {
    RoutePlanner routePlanner;

    public void deliver(Order order) {
        // Calculate route using RoutePlanner
        String route = routePlanner.calculateRoute(order.getSeller().getAddress(), order.get
        System.out.println("Delivery
 route: " + route);

        // Decrease quantity and increase reward
        for (Fruit fruit : order.getFruits()) {
            fruit.setTotalQuantityKg(fruit.getTotalQuantityKg() - order.getQuantityInKg());
        }
        order.getBuyer().setRewards(order.getBuyer().getRewards() + (int) (order.getQuantity
    }
}

class NoFruitsException extends Exception {
    public NoFruitsException(String message) {
        super(message);
    }
}
```

Use code with caution.

Explanation:

1. **Classes:** We've defined the basic classes: `User` , `Fruit` , `Seller` , `Buyer` , `Order` , and `Delivery` .
2. **RoutePlanner Interface:** This interface outlines the `calculateRoute` method, which will be implemented by a specific routing algorithm.
3. **Buyer's `placeOrder` Method:** This method displays the seller's fruits, takes user input, and creates an `Order` object. It also handles the `NoFruitsException` if necessary.
4. **Delivery's `deliver` Method:** This method uses the `RoutePlanner` to calculate the route, updates the `Fruit` quantities, and increases the `Buyer` 's rewards.

Additional Considerations:

- **Error Handling:** Implement robust error handling for invalid inputs, out-of-stock items, and other potential issues.
- **User Interface:** Consider using a GUI or command-line interface to provide a user-friendly experience.

- **Database Integration:** Integrate a database to store user, seller, fruit, and order information.
- **Payment System:** Implement a payment system to handle transactions.
- **Security:** Ensure data security by implementing appropriate measures.
- **Scalability:** Design the system to handle a large number of users, sellers, and orders.

By following these guidelines and incorporating the provided code, you can create a robust and efficient fruit delivery system.