សាកលវិទ្យាល័យភូមិន្ទភ្នំពេញ
មហាវិទ្យាល័យវិទ្យាសាស្ត្រ
ដេប៉ាតឺម៉ង់: ព័ត៌មានវិទ្យា

កិច្ចការស្រាវជ្រាវអំពី: ប្រព័ន្ធគ្រប់គ្រងម៉ាត

ណែនាំដោយសាស្ត្រាចារ្យ: ដាន វ៉ូនផេង

ស៊ូ ចាន់រ៉ូដែម
ធិន ស៊ីវធាន
អន ភក្តី
លិ ស្រីម៉ា

2025~2026

អារម្ភកថា

សូរស្តីប្រិយមិត្តអ្នកអាន ជាទីគាប់ចិត្តទាំងអស់គ្នា សៀវភៅនេះគឺជាសៀវភៅមួយក្បាលដែល និយាយអំពី ប្រព័ន្ធគ្រប់គ្រងផ្សារ (Mart Management System) ដែលត្រូវបានបង្កើតឡើងដើម្បី ជួយសម្រួលដល់ការគ្រប់គ្រងទិន្នន័យ និងសកម្មភាពផ្សេងៗនៅក្នុងសហគ្រាស ឬហាងលក់ ទំនិញ។ ក្នុងសម័យបច្ចុប្បន្ននេះ ដែលបច្ចេកវិទ្យាកំពុងអភិវឌ្ឍទៅមុខយ៉ាងឆាប់រហ័ស ការប្រើប្រាស់ប្រព័ន្ធគ្រប់គ្រងតាមកុំព្យូទ័រ គឺ មានសារៈសំខាន់ខ្លាំងណាស់សម្រាប់អាជីវកម្ម គ្រប់ប្រភេទ។

ប្រព័ន្ធគ្រប់គ្រងផ្សារនេះ ត្រូវបានរចនាឡើង ដើម្បីឲ្យអាចគ្រប់គ្រងព័ត៌មានទាក់ទងនឹង ផលិតផល អតិថិជន ការលក់ ការទិញ និងការទូទាត់ បានយ៉ាងងាយស្រួល និងមានប្រសិទ្ធភាព។ ការប្រើប្រាស់ប្រព័ន្ធនេះ នឹងជួយកាត់បន្ថយពេលវេលា ការខាតបង់ និងកំហុសក្នុងការកត់ត្រា ដោយជំនួសការងារដែលធ្លាប់ធ្វើដោយដៃឲ្យទៅជារបៀបឌីជីថល។

ខ្ញុំសង្ឃឹមថា សៀវភៅនេះ អាចជួយជាឯកសារយោង និងជាមធ្យោបាយសិក្សាសម្រាប់និស្សិត ឬអ្នកដែលចាប់អារម្មណ៍ក្នុងការសិក្សា និងអភិវឌ្ឍប្រព័ន្ធគ្រប់គ្រងទំនើប។ ប្រសិនបើមានកំហុស ឬចំណុចណាមួយដែលមិនត្រឹមត្រូវ សូមប្រិយមិត្តអ្នកអានជួយផ្តល់មតិយោបល់ ដើម្បីអាចកែលម្អឲ្យប្រសើរឡើងជាបន្តទៀត។

ជាចុងក្រោយនេះ ខ្ញុំសូមថ្លែងអំណរគុណយ៉ាងជ្រាលជ្រៅដល់លោកគ្រូ និងមិត្តរួមក្រុម ទាំងអស់ ដែលបានចំណាយពេលវេលា និងខិតខំប្រឹងប្រែងក្នុងការស្រាវជ្រាវ រចនា និងអភិវឌ្ឍប្រព័ន្ធនេះឲ្យបានប្រសើរឡើង។

ភ្នំពេញ ថ្ងៃទី ២២ ខែ កុម្ភៈ ឆ្នាំ ២០២៦

ស៊ូ ចាន់រ៉ូជែម

មាតិកា

1. Objective (គោលបំណង)

គោលបំណងនៃប្រព័ន្ធ Mart Management System គឺដើម្បីជួយឲ្យម្ចាស់សហគ្រាស ឬអ្នកគ្រប់គ្រងផ្សារ អាចគ្រប់គ្រងទិន្នន័យនានាដែលទាក់ទងនឹងអាជីវកម្មយ៉ាងមានប្រសិទ្ធភាព និងប្រើប្រាស់បានងាយស្រួល។ ប្រព័ន្ធនេះអាចអនុវត្តសកម្មភាពជាច្រើន ដូចជា៖

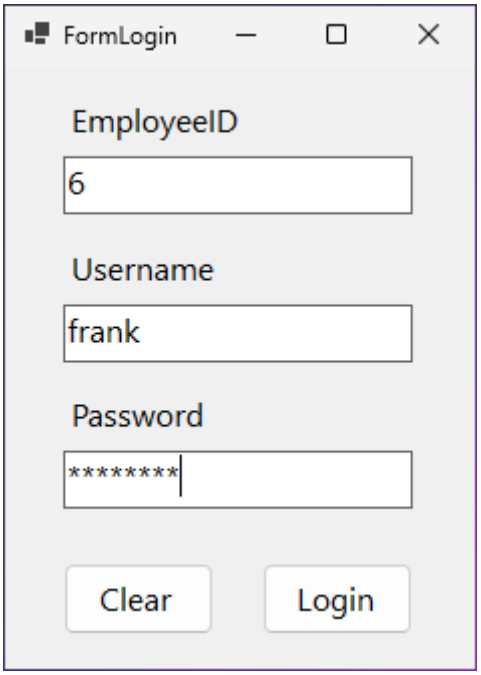
- ការគ្រប់គ្រងព័ត៌មានអំពី ផលិតផល (Products) ដូចជា ឈ្មោះ តម្លៃ ឯកតា និងស្ថានភាពលក់។
- ការគ្រប់គ្រង អតិថិជន (Customers) និង អ្នកផ្គត់ផ្គង់ (Suppliers) ដើម្បីរក្សាទំនាក់ទំនងក្នុងការទិញ និងលក់។
- ការកត់ត្រា ការទិញ (Purchases) និង ការលក់ (Sales) ដើម្បីទាញយករបាយការណ៍ និងតាមដានប្រតិបត្តិការរបស់ហាង។
- ការគ្រប់គ្រង បុគ្គលិក (Employees) និងតួនាទី (Roles) ដើម្បីកំណត់សិទ្ធិប្រើប្រាស់ប្រព័ន្ធ។
- ការគ្រប់គ្រង សារពើភ័ណ្ឌ (Inventory) ដើម្បីដឹងពីបរិមាណទំនិញនៅក្នុងស្តុក និងការបញ្ជាទិញបន្ថែម។
- ការកត់ត្រា ការទូទាត់ (Payments) ដើម្បីតាមដានប្រាក់ចូលចេញ និងស្ថានភាពនៃការបង់ប្រាក់។

ប្រព័ន្ធនេះត្រូវបានរចនាឡើងដោយប្រើ វិធីសាស្ត្រ Object-Oriented Analysis and Design (OOAD) ដែលផ្អែកលើគោលការណ៍ Software Design Pattern ដូចជា *Builder, Singleton, Composite, Decorator, State,* និង *Strategy*។

គោលបំណងចម្បងគឺ៖

- បង្កើតប្រព័ន្ធគ្រប់គ្រងទិន្នន័យដែលមានរចនាសម្ព័ន្ធល្អ និងងាយស្រួលប្រើ។
- បង្ហាញពីការអនុវត្តន៍នៃគោលការណ៍ Software Design Pattern ក្នុងការរចនាប្រព័ន្ធជាក់ស្តែង។
- ជួយឲ្យនិស្សិត ឬអ្នកសិក្សា អាចយល់ពីដំណើរការរចនាប្រព័ន្ធតាម គោលការណ៍វិទ្យាសាស្ត្រ Software Engineering។

2. Login (ចូលប្រើប្រាស់)



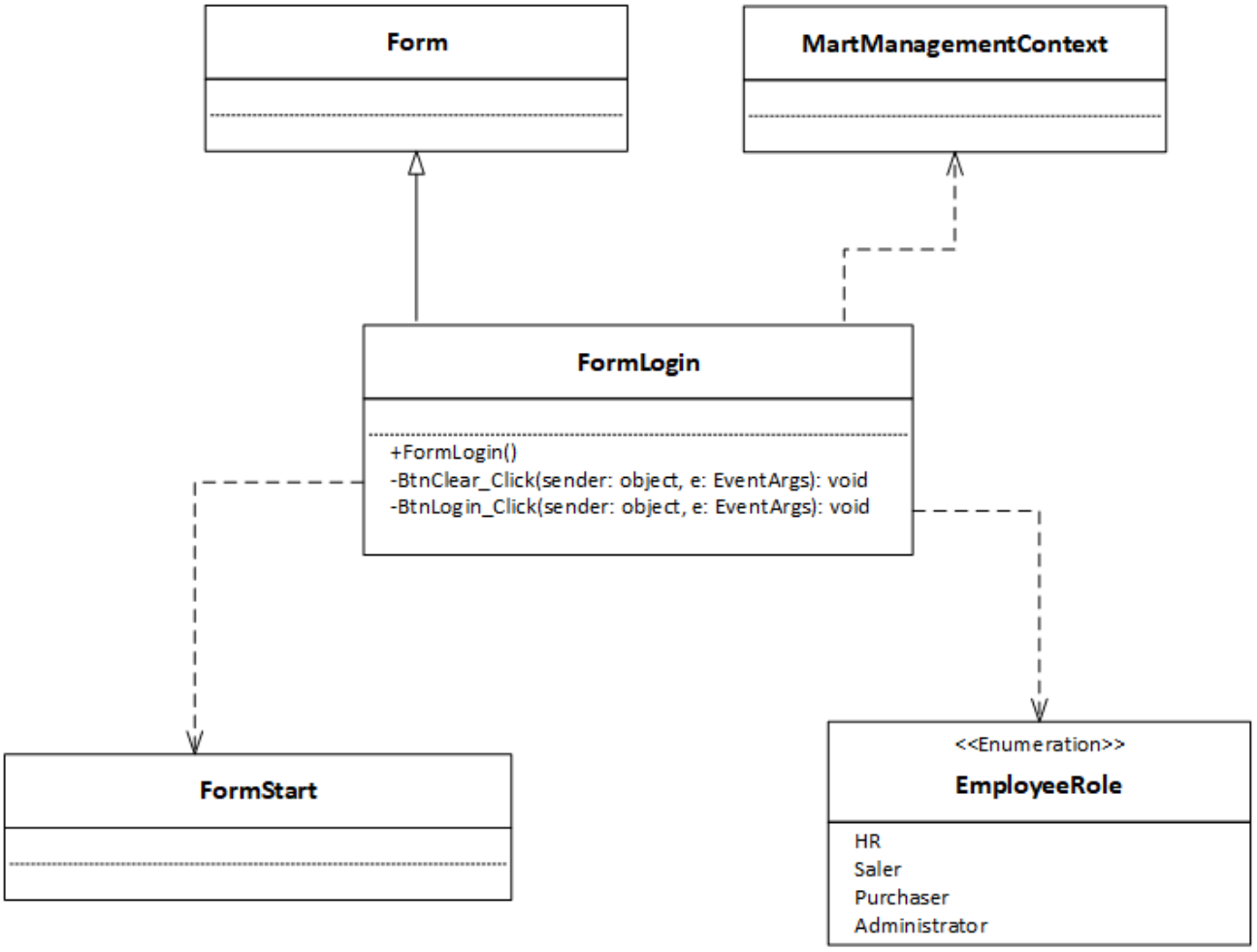
FormLogin គឺជា Form មួយដែលសាកសមសម្រាប់ការផ្ទៀងផ្ទាត់ អត្តសញ្ញាណអ្នកប្រើប្រាស់។

នៅលើផ្ទាំងនេះមាន៖

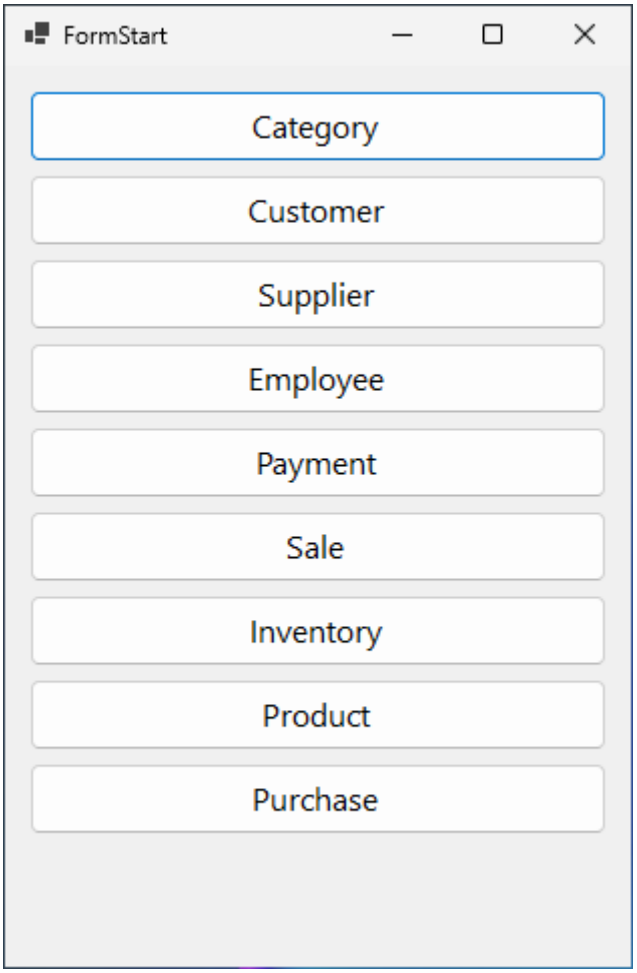
- ប្រអប់បញ្ចូល (TextBox) ចំនួនបី៖
  - EmployeeID – សម្រាប់បញ្ចូលលេខសម្គាល់បុគ្គលិក (ឧ. “6”)
  - Username – សម្រាប់បញ្ចូលឈ្មោះអ្នកប្រើប្រាស់ (ឧ. “frank”)
  - Password – សម្រាប់បញ្ចូលពាក្យសម្ងាត់ ដែលត្រូវបានបិទជាសញ្ញាផ្កាយ (*******)
- ប៊ូតុង (Buttons) ចំនួនពីរ៖
  - Clear – សម្រាប់សម្អាតទិន្នន័យដែលបានបញ្ចូលក្នុងប្រអប់
  - Login – សម្រាប់ចូលប្រព័ន្ធ

Form នេះមានរចនាបថសាមញ្ញ និងមានលក្ខណៈប្រើប្រាស់ងាយស្រួល ។

2.1 UML class diagram



3. Start (ចាប់ផ្តើមប្រើប្រាស់)

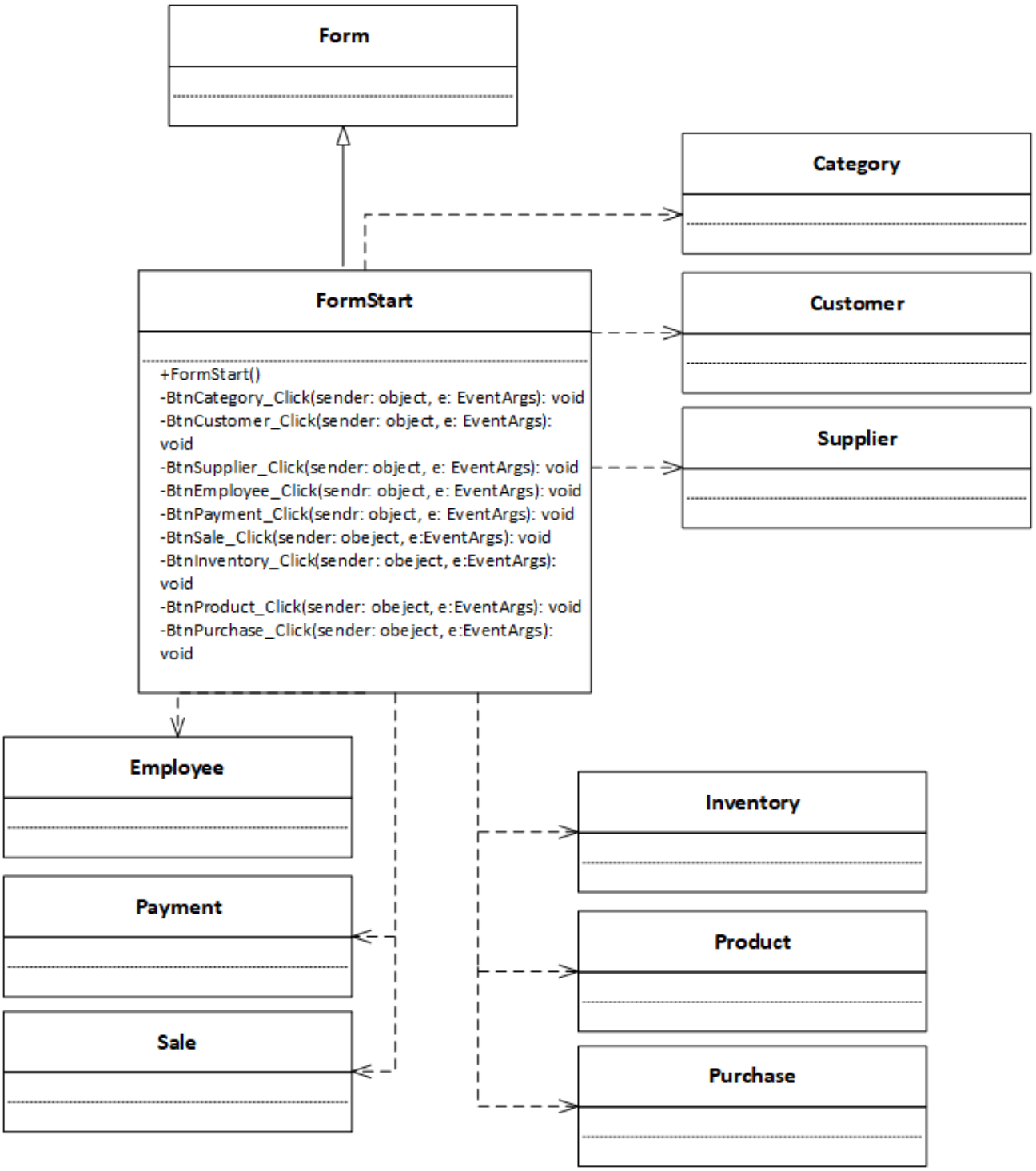


FormStart មានប៊ូតុងចំនួនប្រាំបួន (9) សម្រាប់ចូលទៅកាន់ Form ផ្សេងៗ។
ប៊ូតុងនីមួយៗមានសរសេរចំណងជើងដូចខាងក្រោម៖

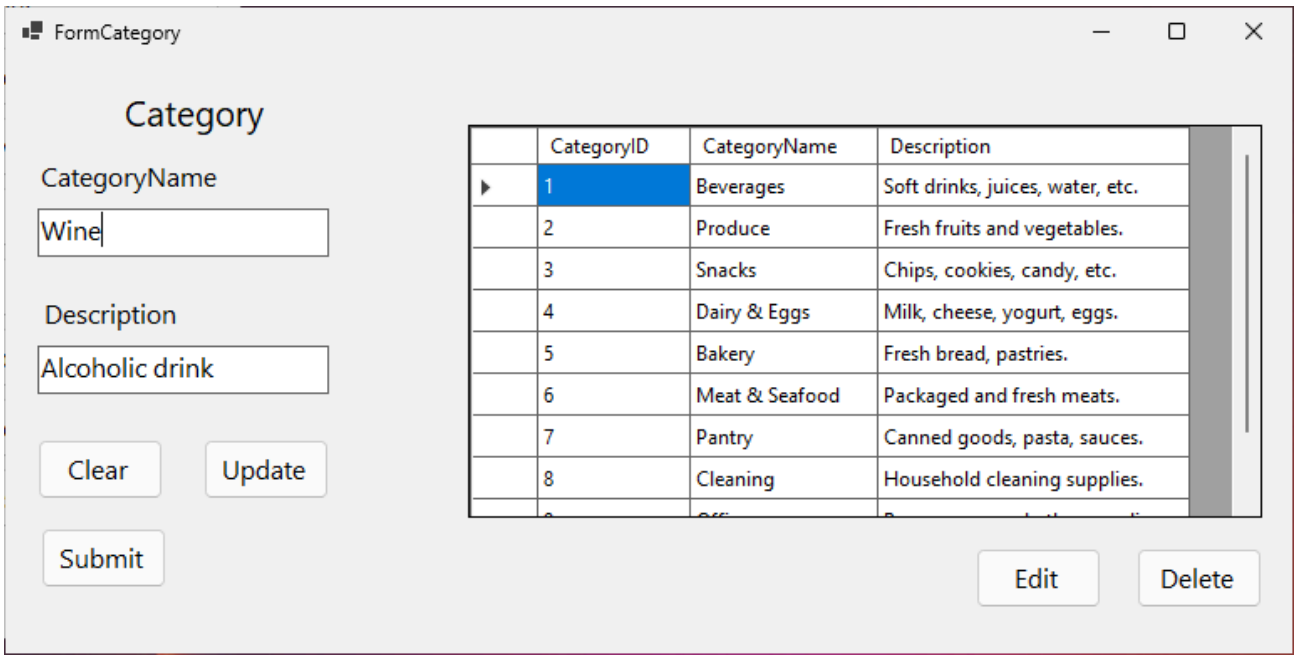
- Category (ប្រភេទ/ចំណាត់ថ្នាក់)
- Customer (អតិថិជន)
- Supplier (អ្នកផ្គត់ផ្គង់)
- Employee (បុគ្គលិក)
- Payment (ការទូទាត់)
- Sale (ការលក់)
- Inventory (សារពើភ័ណ្ឌ/ទំនិញក្នុងស្តុក)
- Product (ផលិតផល)
- Purchase (ការទិញ)

Form នេះជា Main Menu សម្រាប់កម្មវិធីគ្រប់គ្រងអាជីវកម្ម ឬប្រព័ន្ធមូលដ្ឋានទិន្នន័យ។

3.1 UML class diagram



4. Category (ប្រភេទ)



FormCategory នេះ គឺជា Form ដែលត្រូវបានរចនាឡើងសម្រាប់ គ្រប់គ្រងព័ត៌មានប្រភេទទំនិញ (Category Management)។ Form នេះអនុញ្ញាតឲ្យអ្នកប្រើអាចបញ្ចូល កែប្រែ លុប ឬមើលព័ត៌មានអំពីប្រភេទទំនិញ បានយ៉ាងងាយស្រួល។

4.1 ការរៀបចំទម្រង់

4.1.1 ផ្នែកខាងឆ្វេង – ផ្នែកបញ្ចូលទិន្នន័យ

ផ្នែកនេះមានប្រអប់បញ្ចូល (TextBox) និងប៊ូតុងសម្រាប់ធ្វើសកម្មភាពនានា៖

- CategoryName: ប្រអប់សម្រាប់បញ្ចូលឈ្មោះប្រភេទទំនិញ។
  ឧទាហរណ៍៖ *Wine*
- Description: ប្រអប់សម្រាប់សរសេរព័ត៌មានពិពណ៌នាអំពីប្រភេទនោះ។
  ឧទាហរណ៍៖ *Alcoholic drink*

ប៊ូតុងនៅផ្នែកខាងក្រោមរួមមាន៖

- Clear: សម្អាតទិន្នន័យដែលបានបញ្ចូលក្នុងប្រអប់ទាំងអស់។
- Update: កែប្រែទិន្នន័យប្រភេទទំនិញដែលបានជ្រើសរើស។
- Submit: បញ្ចូលប្រភេទទំនិញថ្មីចូលក្នុងប្រព័ន្ធ។

4.1.2 ផ្នែកខាងស្តាំ – តារាងបង្ហាញទិន្នន័យ

ផ្នែកនេះបង្ហាញ តារាង (DataGridView) ដែលផ្ទុកព័ត៌មានអំពីប្រភេទទំនិញទាំងអស់។
តារាងនេះមានជួរឈរ (columns) ៣៖

1. CategoryID – លេខសម្គាល់ប្រភេទទំនិញ
2. CategoryName – ឈ្មោះប្រភេទទំនិញ
3. Description – ពិពណ៌នាប្រភេទទំនិញ

ឧទាហរណ៍ទិន្នន័យដែលមានក្នុងតារាង៖

- Beverages – Soft drinks, juices, water, etc.
- Produce – Fresh fruits and vegetables.
- Snacks – Chips, cookies, candy, etc.
- Dairy & Eggs – Milk, cheese, yogurt, eggs.
- Bakery – Fresh bread, pastries.

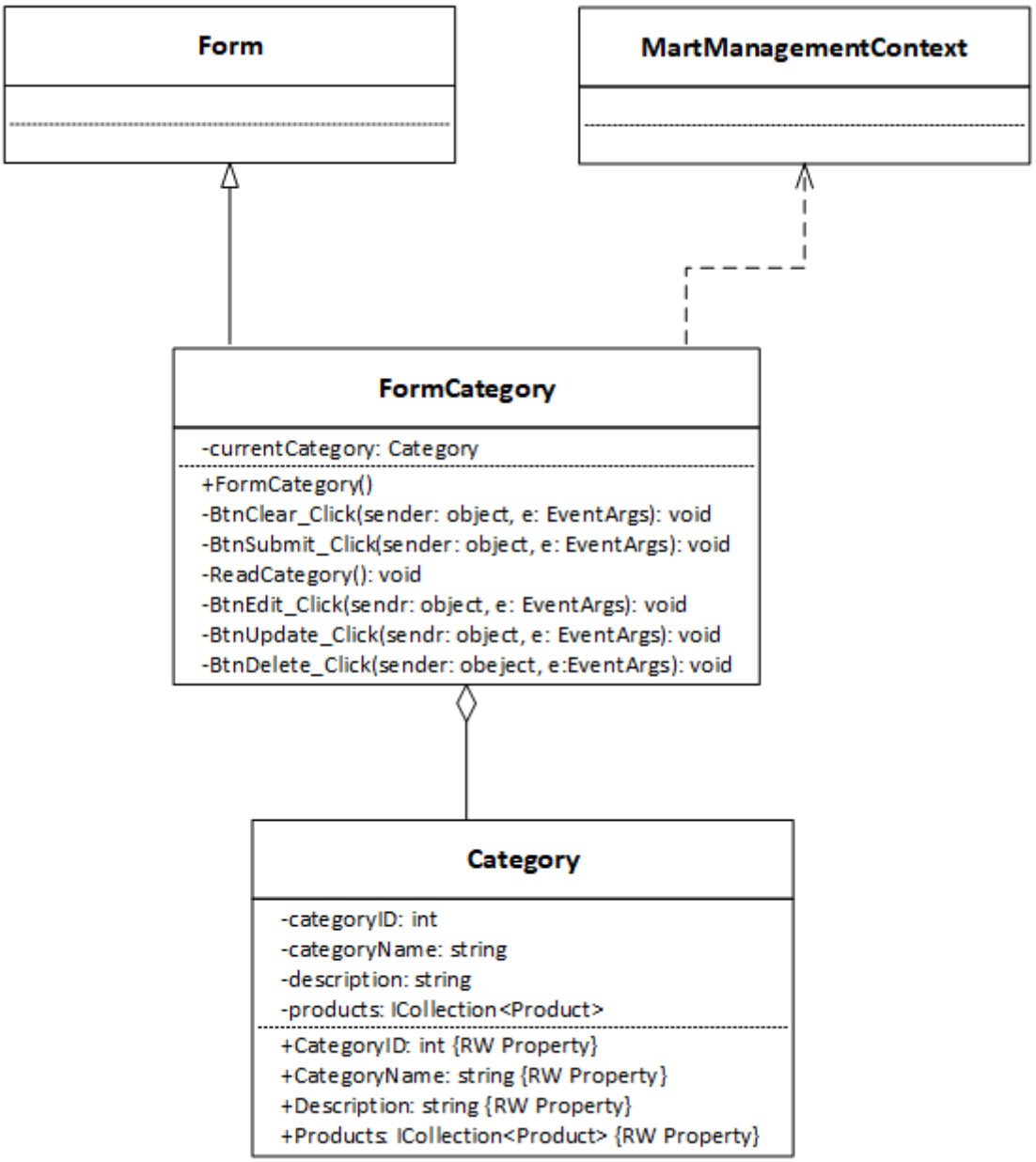
ក្រោមតារាង មានប៊ូតុងបន្ថែម៖

- Edit: ដើម្បីជ្រើសទិន្នន័យមួយក្នុងតារាង ហើយបង្ហាញទៅក្នុងប្រអប់បញ្ចូលសម្រាប់កែប្រែ។
- Delete: ដើម្បីលុបព័ត៌មានប្រភេទទំនិញដែលបានជ្រើសចេញពីប្រព័ន្ធ។

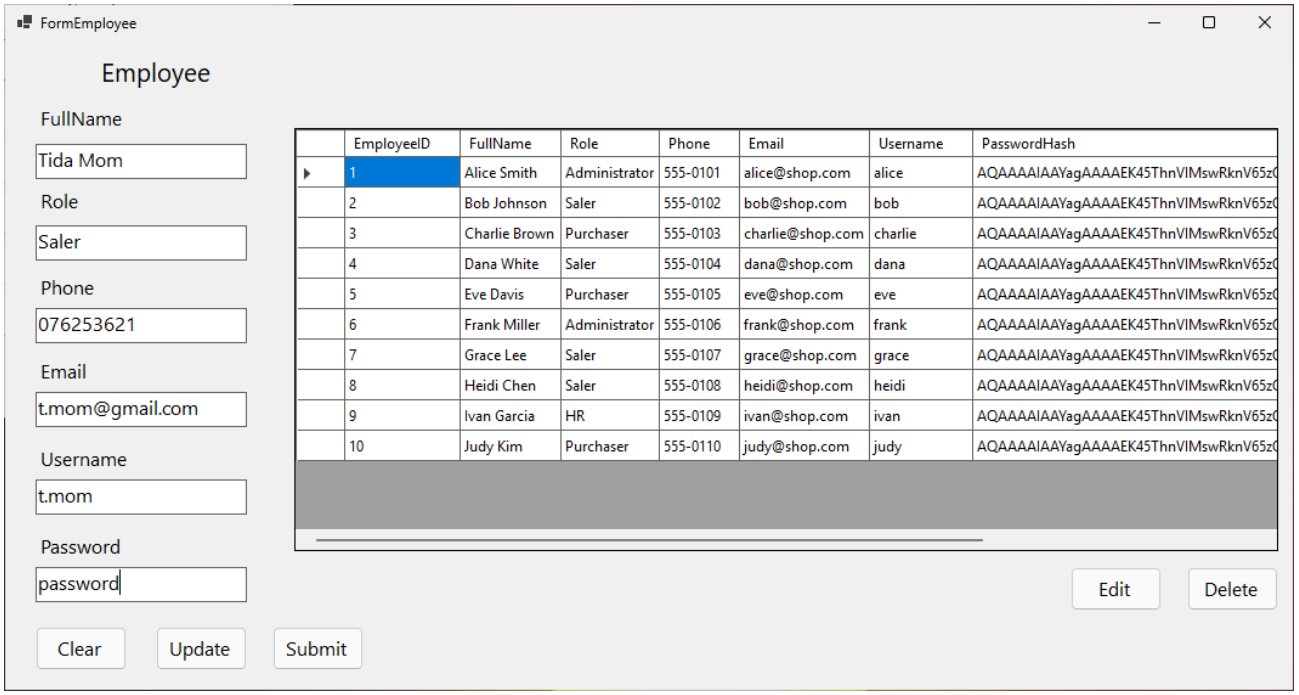
4.2 មុខងារសំខាន់ៗ

ទម្រង់នេះអាចធ្វើសកម្មភាព CRUD (Create, Read, Update, Delete) បានពេញលេញ
ដោយភ្ជាប់ជាមួយមូលដ្ឋានទិន្នន័យ (Database) វាជួយឲ្យអ្នកប្រើអាចគ្រប់គ្រង
ប្រភេទទំនិញបានយ៉ាងមានប្រសិទ្ធភាព និងងាយស្រួល

4.3 UML class diagram



5. Employee (និយោជិត)



FormEmployee អនុញ្ញាតឱ្យអ្នកប្រើមើល ថែម កែប្រែ និងលុបព័ត៌មានបុគ្គលិក។

5.1 ផ្នែកបញ្ចូលទិន្នន័យ (Data Input Section - ខាងឆ្វេង)

Form នេះមានប្រអប់បញ្ចូលព័ត៌មានបុគ្គលិកដូចខាងក្រោម៖

- FullName (ឈ្មោះពេញ): បញ្ចូល Tida Mom
- Role (តួនាទី): បញ្ចូល Saler (អ្នកលក់)
- Phone (ទូរស័ព្ទ): បញ្ចូល 076253621
- Email (អីុមែល): បញ្ចូល t.mom@gmail.com
- Username (ឈ្មោះអ្នកប្រើប្រាស់): បញ្ចូល t.mom
- Password (ពាក្យសម្ងាត់): បញ្ចូល password

ប៊ូតុងដែលមាននៅផ្នែកខាងក្រោមនៃការបញ្ចូលទិន្នន័យគឺ៖

- Clear (ជម្រះ): សម្រាប់ជម្រះទិន្នន័យពីប្រអប់ទាំងអស់។
- Update (ធ្វើបច្ចុប្បន្នភាព): សម្រាប់កែប្រែព័ត៌មានបុគ្គលិកដែលមានស្រាប់។
- Submit (ដាក់ស្នើ): សម្រាប់បញ្ចូលព័ត៌មានបុគ្គលិកថ្មី។

5.2 ផ្នែកតារាងទិន្នន័យ (Data Table Section - ខាងស្តាំ)

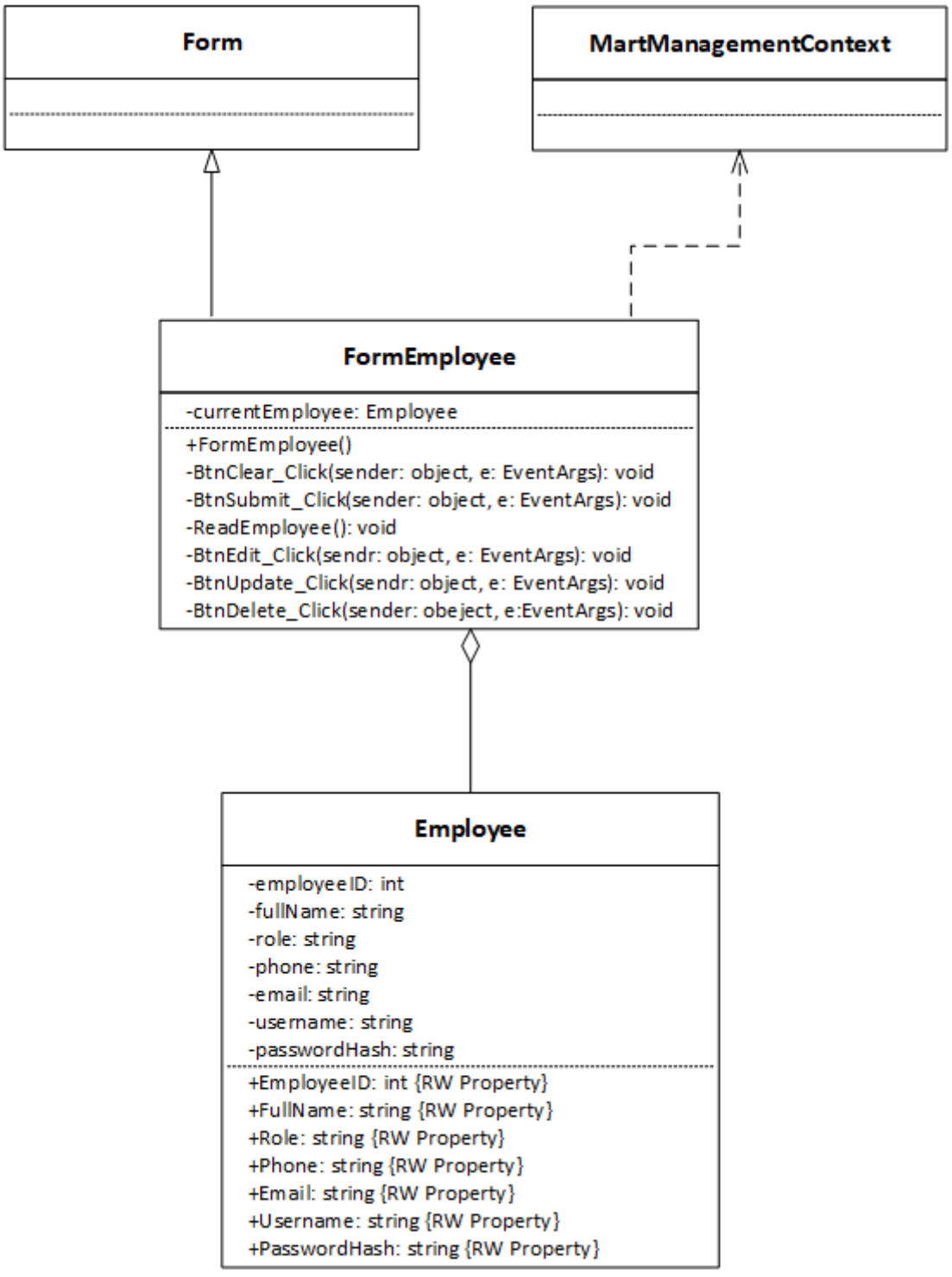
ផ្នែកខាងស្តាំបង្ហាញ តារាងបញ្ជីបុគ្គលិក ចំនួន ១០ នាក់ ដែលមានព័ត៌មានដូចជា៖

- EmployeeID (លេខសម្គាល់បុគ្គលិក): លេខរៀងពី ១ ដល់ ១០។
- FullName (ឈ្មោះពេញ): ដូចជា Alice Smith, Bob Johnson, Charlie Brown ។ល។
- Role (តួនាទី): ដូចជា Administrator, Saler, Purchaser, HR ។ល។
- Phone (ទូរស័ព្ទ)
- Email (អ៊ីមែល)
- Username (ឈ្មោះអ្នកប្រើប្រាស់)
- PasswordHash (កូដពាក្យសម្ងាត់): ជាខ្សែអក្សរវែងដែលបានអ៊ិនគ្រីប (Encrypted String) ដើម្បីសុវត្ថិភាព។

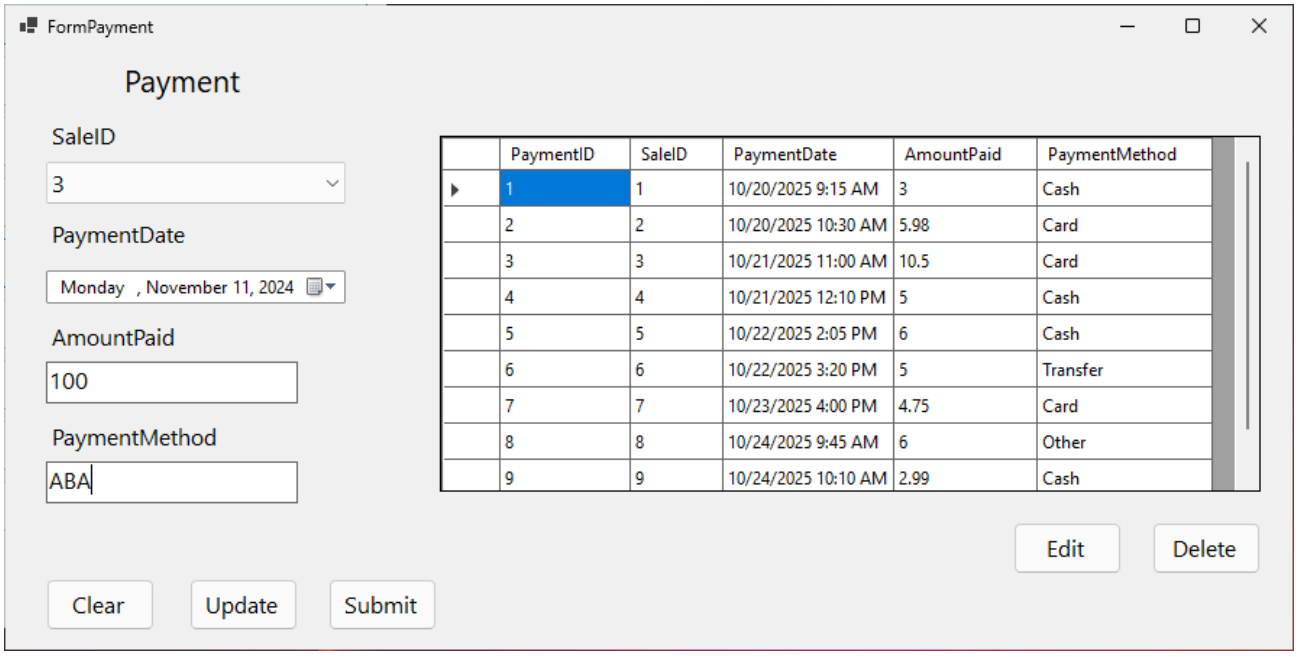
ប៊ូតុងដែលមាននៅក្រោមតារាងគឺ៖

- Edit (កែប្រែ):
  ទំនងជាសម្រាប់ផ្ទុកទិន្នន័យពីជួរដែលបានជ្រើសរើសទៅក្នុងប្រអប់បញ្ចូលខាងឆ្វេង។
- Delete (លុប): សម្រាប់លុបកំណត់ត្រាបុគ្គលិកដែលបានជ្រើសរើសចេញពីប្រព័ន្ធ។

5.3 UML class diagram



6. Payment (ការទូទាត់)



Form ការទូទាត់ប្រាក់ (FormPayment) ជាកន្លែងសម្រាប់កត់ត្រា និងគ្រប់គ្រងប្រតិបត្តិការទូទាត់នានា។

6.1 ផ្នែកបញ្ចូលទិន្នន័យ (Data Input Section - ខាងឆ្វេង)

Form នេះមានប្រអប់បញ្ចូលព័ត៌មានសំខាន់ៗសម្រាប់ការទូទាត់មួយលើកៗ៖

- SaleID (លេខសម្គាល់ការលក់): បង្ហាញលេខ 3 (ជាទូទៅ គឺសម្រាប់ភ្ជាប់ការទូទាត់នេះទៅនឹងវិក្កយបត្រលក់ជាក់លាក់ណាមួយ)។
- PaymentDate (កាលបរិច្ឆេទទូទាត់): បង្ហាញ Monday, November 11, 2024។
- AmountPaid (ចំនួនទឹកប្រាក់ដែលបានបង់): បង្ហាញ 100 (ដុល្លារឬរៀលក៏ផ្សេង)។
- PaymentMethod (វិធីទូទាត់): បង្ហាញ ABA (ឈ្មោះធនាគារទូទាត់តាម អេឡិចត្រូនិក)។

ប៊ូតុងដែលមាននៅផ្នែកខាងក្រោមនៃការបញ្ចូលទិន្នន័យគឺ៖

- Clear (ជម្រះ): សម្រាប់ជម្រះទិន្នន័យពីប្រអប់ទាំងអស់។
- Update (ធ្វើបច្ចុប្បន្នភាព): សម្រាប់កែប្រែកំណត់ត្រាទូទាត់ដែលមានស្រាប់។
- Submit (ដាក់ស្នើ): សម្រាប់បញ្ចូលកំណត់ត្រាទូទាត់ថ្មី។

6.2 ផ្នែកតារាងទិន្នន័យ (Data Table Section - ខាងស្តាំ)

ផ្នែកខាងស្តាំបង្ហាញ តារាងបញ្ជីប្រតិបត្តិការទូទាត់ ចំនួន ៩ លើក ដោយមានព័ត៌មានលម្អិតដូចជា៖

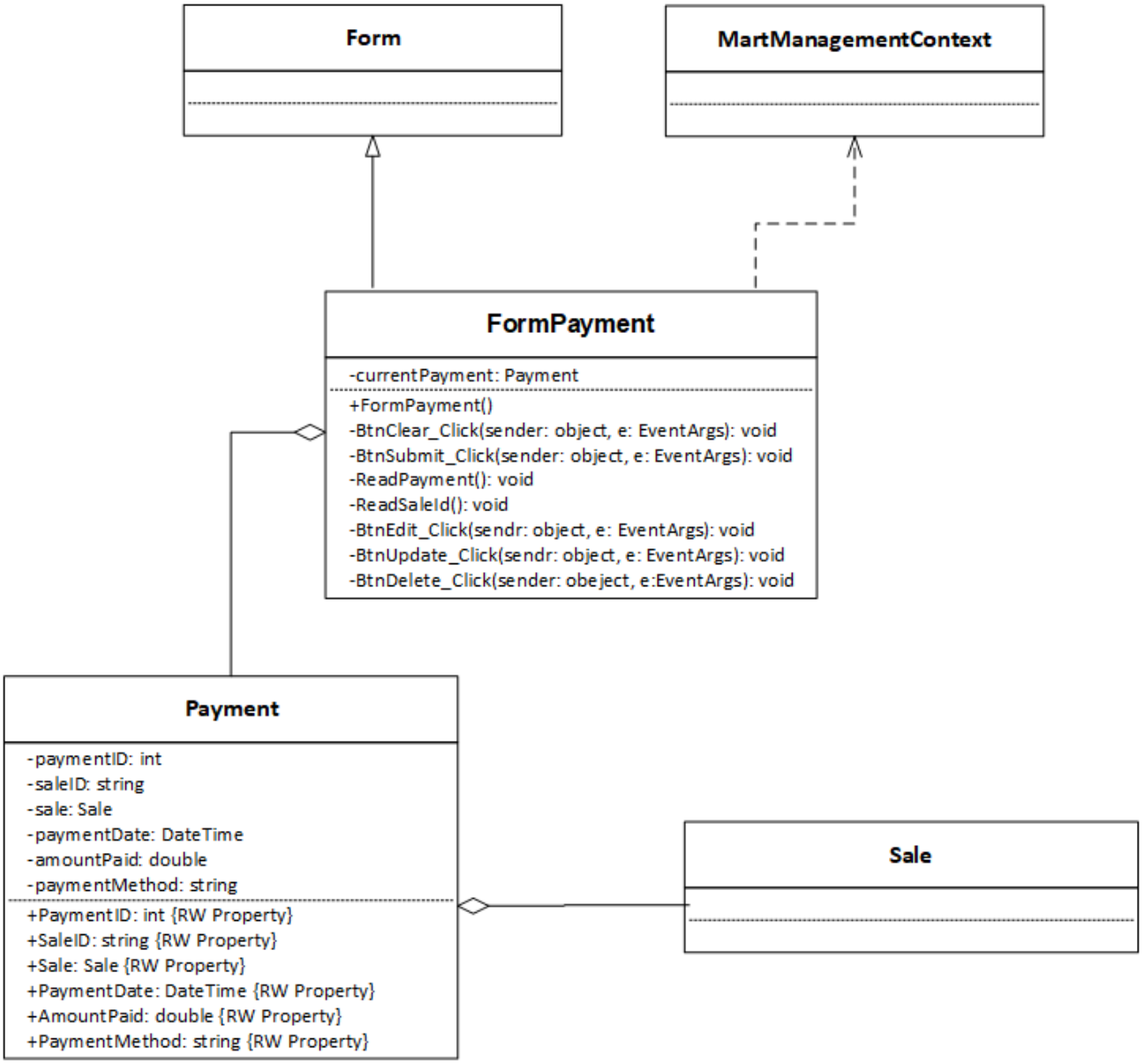
- PaymentID (លេខសម្គាល់ការទូទាត់): លេខរៀងពី ១ ដល់ ៩។

- SaleID (លេខសម្គាល់ការលក់): លេខសម្រាប់ភ្ជាប់ទៅនឹងការលក់។
- PaymentDate (កាលបរិច្ឆេទទូទាត់): រួមទាំងម៉ោង (ឧទាហរណ៍ 10/20/2025 9:15 AM)។
- AmountPaid (ចំនួនទឹកប្រាក់ដែលបានបង់): ដូចជា 3, 5.98, 10.5, 100 ។ល។
- PaymentMethod (វិធីទូទាត់): មាន Cash (សាច់ប្រាក់), Card (កាត), Transfer (ផ្ទេរ) និង Other (ផ្សេងៗ)។

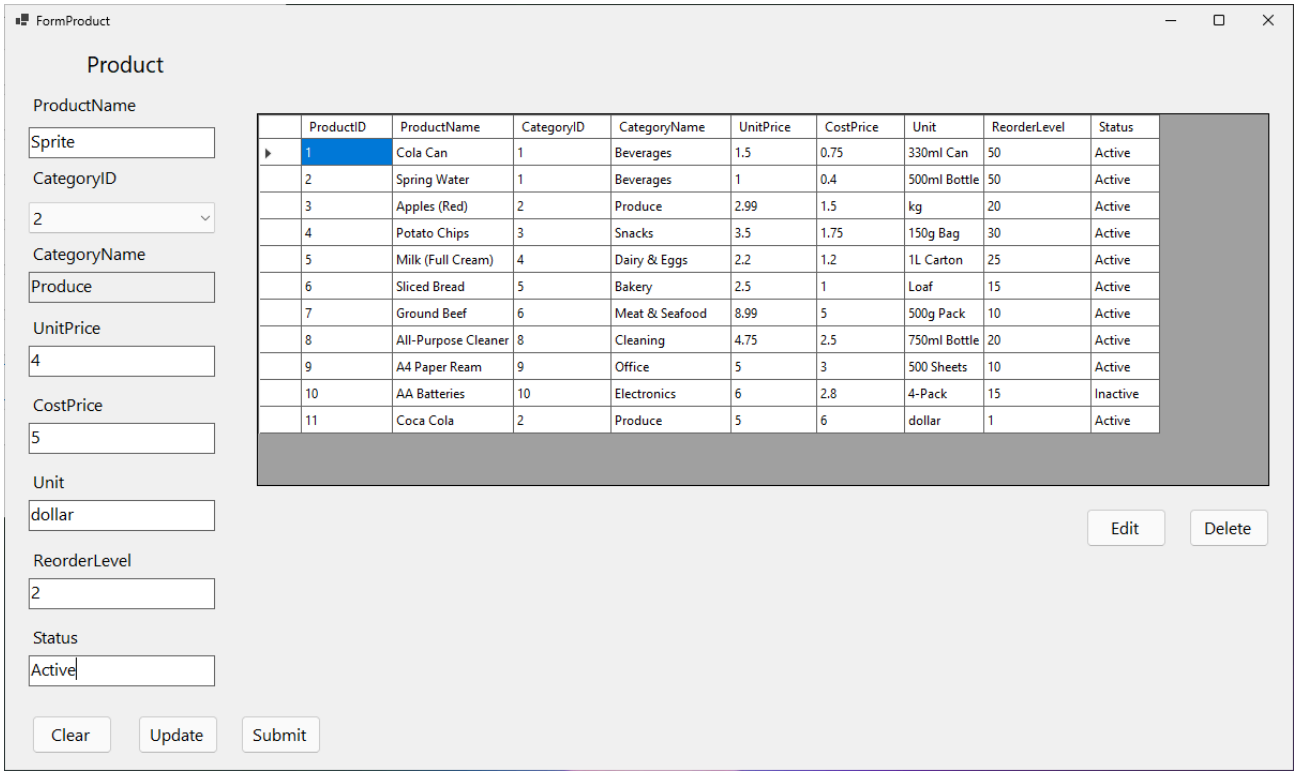
ប៊ូតុងដែលមាននៅក្រោមតារាងគឺ៖

- Edit (កែប្រែ): សម្រាប់ផ្ទុកទិន្នន័យពីជួរដែលបានជ្រើសរើសទៅក្នុងប្រអប់ បញ្ចូលខាងឆ្វេង។
- Delete (លុប): សម្រាប់លុបកំណត់ត្រាទូទាត់ដែលបានជ្រើសរើស។

6.3 UML class diagram



7. Product (ផលិតផល)



FormProduct អនុញ្ញាតឱ្យអ្នកប្រើមើល ថែម កែប្រែ និងលុបព័ត៌មានលម្អិតរបស់ទំនិញ ឬផលិតផលទាំងអស់។

7.1 ផ្នែកបញ្ចូលទិន្នន័យ (Data Input Section - ខាងឆ្វេង)

ទម្រង់នេះមានប្រអប់បញ្ចូលព័ត៌មានសំខាន់ៗសម្រាប់ផលិតផលនីមួយៗ៖

- ProductName (ឈ្មោះផលិតផល): បញ្ចូល Sprite
- CategoryID (លេខសម្គាល់ប្រភេទ): បញ្ចូល 2
- CategoryName (ឈ្មោះប្រភេទ): បញ្ចូល Produce (ផលិតផលកសិកម្ម)
- UnitPrice (តម្លៃលក់): បញ្ចូល 4 (តម្លៃដែលក្រុមហ៊ុនលក់ចេញ)
- CostPrice (តម្លៃដើម): បញ្ចូល 5 (តម្លៃដែលក្រុមហ៊ុនទិញចូល)
- Unit (ឯកតា): បញ្ចូល dollar (ឯកតាសម្រាប់ថ្លៃ/តម្លៃ)
- ReorderLevel (កម្រិតត្រូវកម៉្ង់បន្ថែម): បញ្ចូល 2 (កម្រិតស្តុកអប្បបរមាដែលទាមទារឱ្យមានការទិញចូលបន្ថែម)
- Status (ស្ថានភាព): បញ្ចូល Active (បង្ហាញថាផលិតផលនេះកំពុងធ្វើអាជីវកម្ម)

ប៊ូតុងដែលមាននៅផ្នែកខាងក្រោមនៃការបញ្ចូលទិន្នន័យគឺ៖

- Clear (ជម្រះ): សម្រាប់ជម្រះទិន្នន័យពីប្រអប់ទាំងអស់។

- Update (ធ្វើបច្ចុប្បន្នភាព): សម្រាប់កែប្រែព័ត៌មានផលិតផលដែលមានស្រាប់។
- Submit (ដាក់ស្នើ): សម្រាប់បញ្ចូលកំណត់ត្រាផលិតផលថ្មី។

7.2 ផ្នែកតារាងទិន្នន័យ (Data Table Section - ខាងស្តាំ)

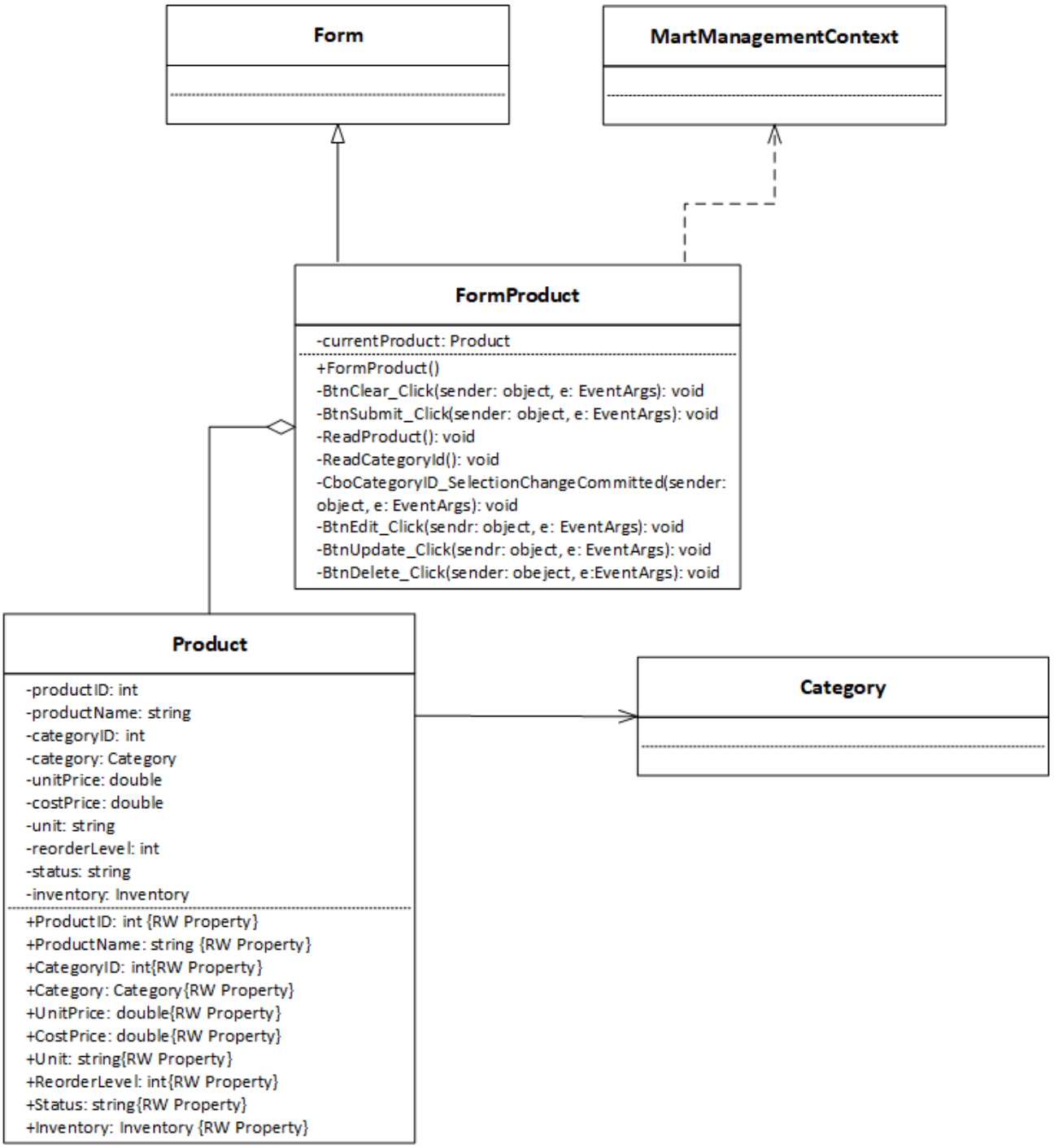
ផ្នែកខាងស្តាំបង្ហាញ តារាងបញ្ជីផលិតផល ចំនួន ១១ មុខ ដោយមានព័ត៌មានលម្អិតសំខាន់ៗ៖

- ProductID (លេខសម្គាល់ផលិតផល)
- ProductName (ឈ្មោះផលិតផល): ដូចជា Cola Can, Spring Water, Apples (Red) ។ល។
- CategoryID (លេខសម្គាល់ប្រភេទ)
- CategoryName (ឈ្មោះប្រភេទ): ដូចជា Beverages, Produce, Snacks ។ល។
- UnitPrice (តម្លៃលក់)
- CostPrice (តម្លៃដើម)
- Unit (ឯកតា): ដូចជា 330ml Can, kg, Loaf, dollar ។ល។
- ReorderLevel (កម្រិតត្រូវកម្ម៉ង់បន្ថែម)
- Status (ស្ថានភាព): ភាគច្រើន Active (កំពុងលក់) និងមួយមុខគឺ Inactive (មិនលក់)។

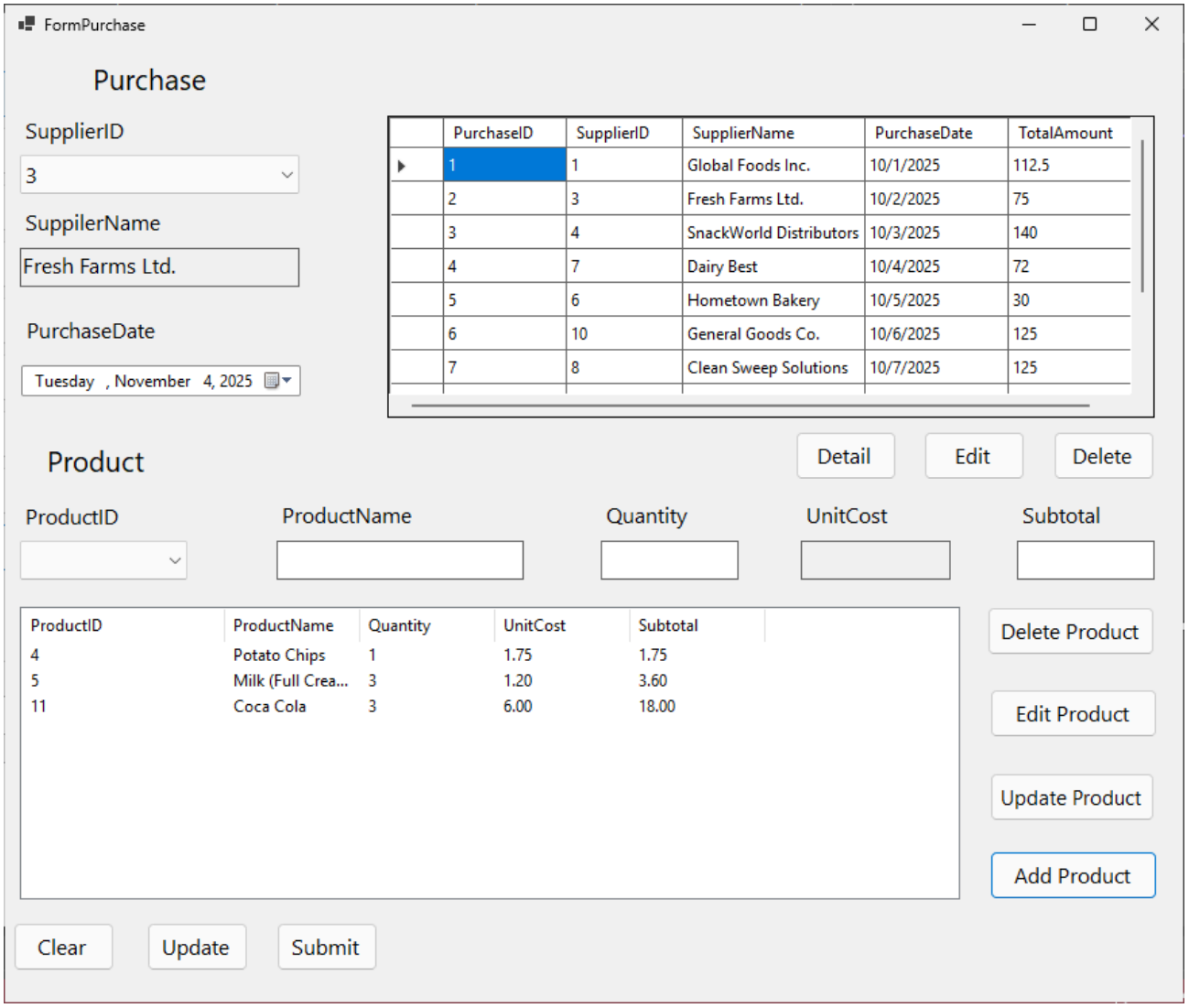
ប៊ូតុងដែលមាននៅក្រោមតារាងគឺ៖

- Edit (កែប្រែ): សម្រាប់ផ្ទុកទិន្នន័យពីជួរដែលបានជ្រើសរើសទៅក្នុងប្រអប់បញ្ចូលខាងឆ្វេង។
- Delete (លុប): សម្រាប់លុបកំណត់ត្រាផលិតផលដែលបានជ្រើសរើស។

7.3 UML class diagram



8. Purchase (ទិញ)



FormPurchase ប្រើសម្រាប់កត់ត្រារាល់ប្រតិបត្តិការទិញទំនិញ ឬវត្ថុធាតុដើមពីអ្នកផ្គត់ផ្គង់ (Suppliers)។

Form នេះចែកចេញជាបីផ្នែកសំខាន់ៗ៖ ព័ត៌មានទិញទូទៅ (Header), បញ្ជីការទិញដែលមាន ស្រាប់ (Purchase History), និងព័ត៌មានលម្អិតផលិតផលដែលបានទិញ (Details)។

8.1 ព័ត៌មានទិញទូទៅ (Purchase Header - ផ្នែកខាងលើឆ្វេង)

- SupplierID (លេខសម្គាល់អ្នកផ្គត់ផ្គង់): បង្ហាញលេខ 3 (សម្រាប់កំណត់អត្តសញ្ញាណក្រុមហ៊ុនដែលយើងទិញទំនិញពី)។
- SupplierName (ឈ្មោះអ្នកផ្គត់ផ្គង់): បង្ហាញ Fresh Farms Ltd.
- PurchaseDate (កាលបរិច្ឆេទទិញ): បង្ហាញ Tuesday, November 4, 2025។

8.2 តារាងបញ្ជីការទិញ (Purchase History Table - ផ្នែកខាងលើស្តាំ)

តារាងនេះបង្ហាញប្រវត្តិនៃការទិញចូលដែលបានធ្វើរួចហើយ ដោយមានជួរឈរដូចជា៖

- PurchaseID (លេខសម្គាល់ការទិញ)
- SupplierID (លេខសម្គាល់អ្នកផ្គត់ផ្គង់)
- SupplierName (ឈ្មោះអ្នកផ្គត់ផ្គង់)
- PurchaseDate (កាលបរិច្ឆេទទិញ)
- TotalAmount (តម្លៃសរុប): ចំនួនទឹកប្រាក់សរុបនៃការទិញនោះ។

ប៊ូតុងដែលទាក់ទងនឹងតារាងនេះគឺ៖

- Detail (លម្អិត):
  ទំនងជាសម្រាប់មើលព័ត៌មានផលិតផលលម្អិតនៃការទិញដែលបានជ្រើសរើស។
- Edit (កែប្រែ)
- Delete (លុប)

8.3 ព័ត៌មានលម្អិតផលិតផល (Product Details - ផ្នែកខាងក្រោម)

ផ្នែកនេះប្រើសម្រាប់បញ្ចូលផលិតផលនីមួយៗទៅក្នុងវិក្កយបត្រទិញចូល៖

ក. ប្រអប់បញ្ចូលផលិតផល:

- ProductID (លេខសម្គាល់ផលិតផល)
- ProductName (ឈ្មោះផលិតផល)
- Quantity (បរិមាណ)
- UnitCost (តម្លៃក្នុងមួយឯកតា)
- Subtotal (តម្លៃរង): តម្លៃសរុបសម្រាប់ផលិតផលនោះ (Quantity x UnitCost)។

ខ. តារាងផលិតផលលម្អិត (Product List):

- បង្ហាញផលិតផលដែលបានបញ្ចូលរួចហើយក្នុងការទិញបច្ចុប្បន្ន (ឧទាហរណ៍ Potato Chips, Milk, Coca Cola) រួមជាមួយ Quantity, UnitCost និង Subtotal។

គ. ប៊ូតុងប្រតិបត្តិការផលិតផល:

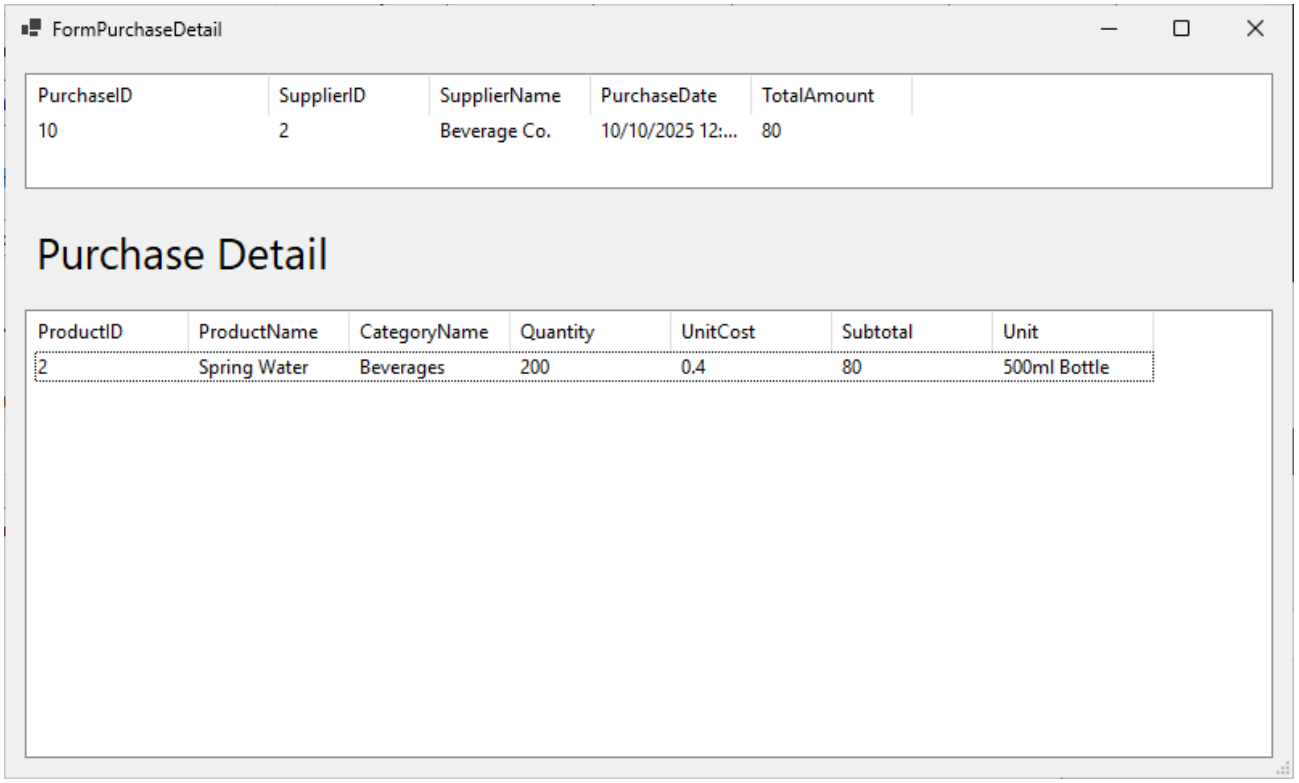
- Delete Product (លុបផលិតផល): លុបផលិតផលដែលបានជ្រើសរើសពីតារាងលម្អិត។
- Edit Product (កែប្រែផលិតផល)
- Update Product (ធ្វើបច្ចុប្បន្នភាពផលិតផល)

- Add Product (បន្ថែមផលិតផល): បន្ថែមផលិតផលថ្មីទៅក្នុងតារាងលម្អិត។

8.4 ប៊ូតុងបញ្ជូនទិន្នន័យចុងក្រោយ (Main Submission Buttons - ផ្នែកខាងក្រោមឆ្វេង)

- Clear (ជម្រះ): ជម្រះទិន្នន័យទាំងអស់ក្នុងទម្រង់។
- Update (ធ្វើបច្ចុប្បន្នភាព): ធ្វើបច្ចុប្បន្នភាពកំណត់ត្រាការទិញដែលមានស្រាប់។
- Submit (ដាក់ស្នើ): បញ្ជូនការទិញចូលថ្មីទៅក្នុងប្រព័ន្ធ។

8.5 Purchase Detail



FormPurchaseDetail ត្រូវបានហៅចេញពីទម្រង់ការទិញ (FormPurchase) ដើម្បីបង្ហាញពីព័ត៌មានលម្អិតនៃទំនិញដែលបានទិញក្នុងប្រតិបត្តិការនីមួយៗ។

8.5.1 ព័ត៌មានទិញទូទៅ (Purchase Header Information - ផ្នែកខាងលើ)

ផ្នែកនេះបង្ហាញសេចក្តីសង្ខេបអំពីវិក្កយបត្រទិញចូលដែលបានជ្រើសរើស គឺការទិញលេខ 10 (PurchaseID 10)៖

- PurchaseID (លេខសម្គាល់ការទិញ): 10
- SupplierID (លេខសម្គាល់អ្នកផ្គត់ផ្គង់): 2
- SupplierName (ឈ្មោះអ្នកផ្គត់ផ្គង់): Beverage Co.
- PurchaseDate (កាលបរិច្ឆេទទិញ): 10/10/2025

- TotalAmount (តម្លៃសរុប): 80

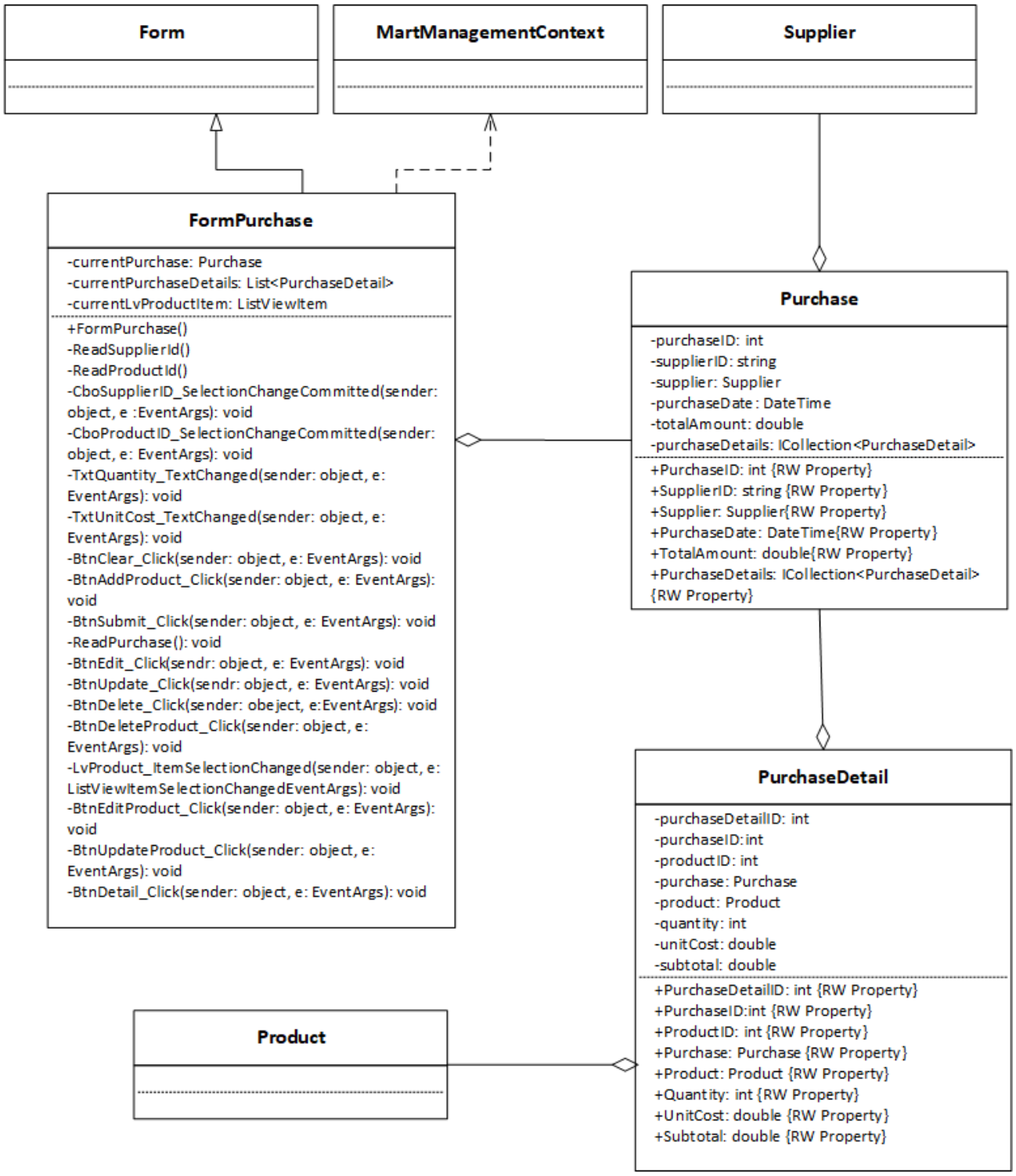
8.5.2 តារាងលម្អិតផលិតផលដែលបានទិញ (Product Detail Table - ផ្នែកកណ្តាល)

នៅក្រោមចំណងជើង "Purchase Detail" (លម្អិតនៃការទិញ)
មានតារាងមួយដែលបង្ហាញពីផលិតផលដែលបានទិញចូលក្នុងប្រតិបត្តិការនេះ។ ក្នុងករណីនេះ
មានផលិតផលតែមួយមុខប៉ុណ្ណោះដែលត្រូវបានកត់ត្រា៖

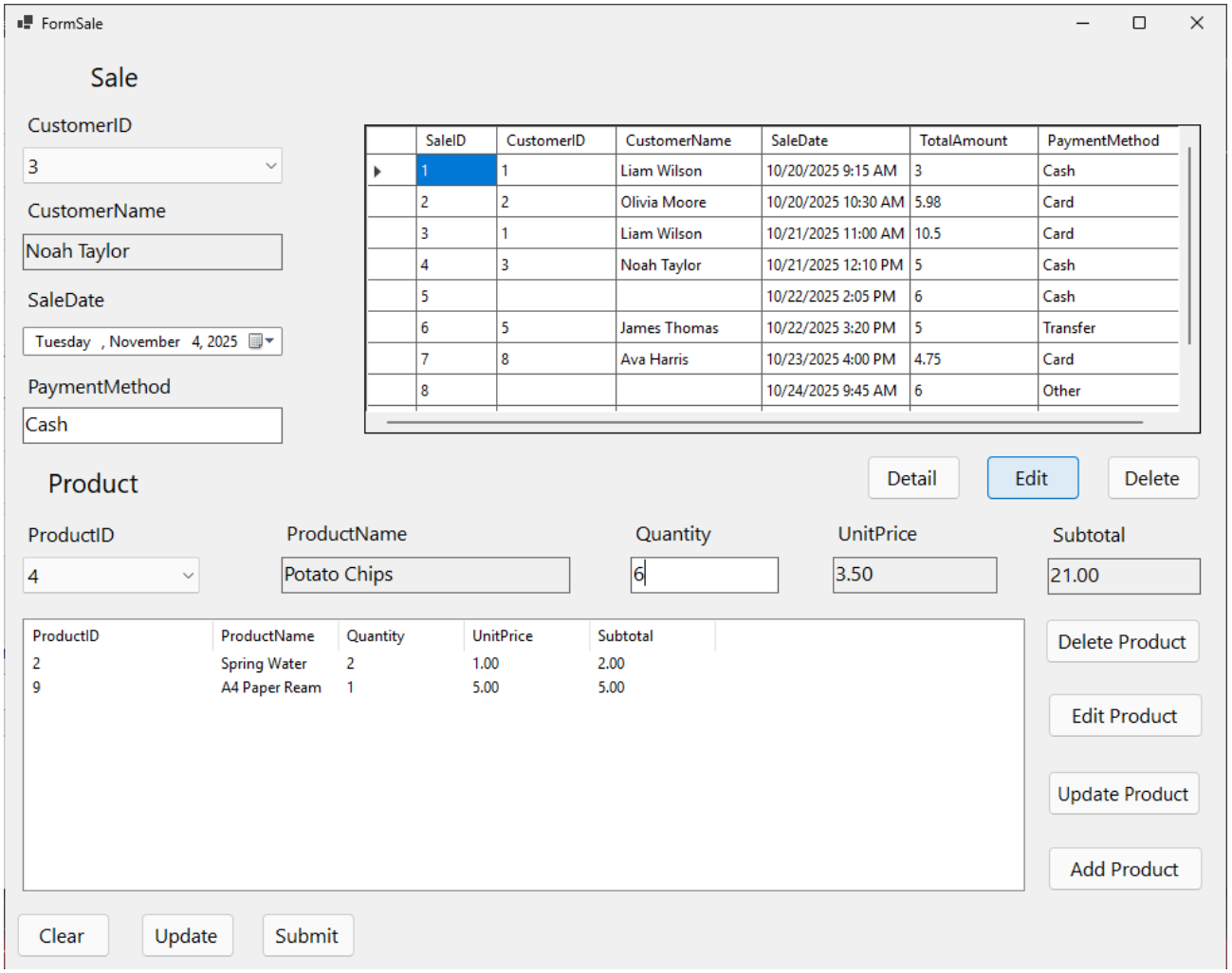
ជួរឈរ (Column)	ព័ត៌មាន (Information)
ProductID (លេខសម្គាល់ផលិតផល)	2
ProductName (ឈ្មោះផលិតផល)	Spring Water (ទឹកបរិសុទ្ធ)
CategoryName (ឈ្មោះប្រភេទ)	Beverages (ភេសជ្ជៈ)
Quantity (បរិមាណ)	200
UnitCost (តម្លៃដើមក្នុងមួយឯកតា)	0.4
Subtotal (តម្លៃរង)	80
Unit (ឯកតា)	500ml Bottle (ដប 500 មីលីលីត្រ)

សរុបមក: ការទិញលេខ 10 ពីក្រុមហ៊ុន Beverage Co. ដែលមានតម្លៃសរុប 80 នេះ គឺជាការទិញ Spring Water ចំនួន 200 ដប (តម្លៃដើម 0.4 ក្នុងមួយដប)។

8.6 UML class diagram



9. Sale (លក់)



FormSale ប្រើសម្រាប់កត់ត្រារាល់ប្រតិបត្តិការលក់ទំនិញ ឬសេវាកម្មទៅឱ្យអតិថិជន (Customers)។

Form នេះមានលក្ខណៈស្រដៀងនឹងទម្រង់ការទិញ (FormPurchase)
គឺត្រូវបានបែងចែកជាបីផ្នែកសំខាន់ៗ៖ ព័ត៌មានលក់ទូទៅ (Header), បញ្ជីការលក់ដែលមានស្រាប់ (Sales History), និងព័ត៌មានលម្អិតផលិតផលដែលបានលក់ (Details)។

9.1 ព័ត៌មានលក់ទូទៅ (Sale Header - ផ្នែកខាងលើឆ្វេង)

- CustomerID (លេខសម្គាល់អតិថិជន): បង្ហាញលេខ 3 (សម្រាប់កំណត់អត្តសញ្ញាណអតិថិជន)។
- CustomerName (ឈ្មោះអតិថិជន): បង្ហាញ Noah Taylor។
- SaleDate (កាលបរិច្ឆេទលក់): បង្ហាញ Tuesday, November 4, 2025។
- PaymentMethod (វិធីទូទាត់): បង្ហាញ Cash (សាច់ប្រាក់)។

9.2 តារាងបញ្ជីការលក់ (Sales History Table - ផ្នែកខាងលើស្តាំ)

តារាងនេះបង្ហាញប្រវត្តិនៃការលក់ដែលបានធ្វើរួចហើយ ចំនួន ៨ លើក ដោយមានជួរឈរដូចជា៖

- SaleID (លេខសម្គាល់ការលក់)
- CustomerID (លេខសម្គាល់អតិថិជន)
- CustomerName (ឈ្មោះអតិថិជន)
- SaleDate (កាលបរិច្ឆេទលក់)
- TotalAmount (តម្លៃសរុប)
- PaymentMethod (វិធីទូទាត់): មាន Cash, Card, Transfer, Other។

ប៊ូតុងដែលទាក់ទងនឹងតារាងនេះគឺ៖

- Detail (លម្អិត): សម្រាប់មើលព័ត៌មានផលិតផលលម្អិតនៃការលក់ដែលបានជ្រើសរើស។
- Edit (កែប្រែ)
- Delete (លុប)

9.3 ព័ត៌មានលម្អិតផលិតផល (Product Details - ផ្នែកខាងក្រោម)

ផ្នែកនេះប្រើសម្រាប់បញ្ចូលផលិតផលនីមួយៗទៅក្នុងវិក្កយបត្រលក់បច្ចុប្បន្ន៖

ក. ប្រអប់បញ្ចូលផលិតផល:

- ProductID (លេខសម្គាល់ផលិតផល): បង្ហាញ 4។
- ProductName (ឈ្មោះផលិតផល): បង្ហាញ Potato Chips។
- Quantity (បរិមាណ): បង្ហាញ 6។
- UnitPrice (តម្លៃលក់ក្នុងមួយឯកតា): បង្ហាញ 3.5។
- Subtotal (តម្លៃរង): បង្ហាញ 21.00 (បានមកពី Quantity 6 x UnitPrice 3.5)។

ខ. តារាងផលិតផលលម្អិត (Product List):

- បង្ហាញផលិតផលដែលបានបញ្ចូលរួចហើយក្នុងការលក់បច្ចុប្បន្ន (ឧទាហរណ៍ Spring Water និង A4 Paper Ream)។

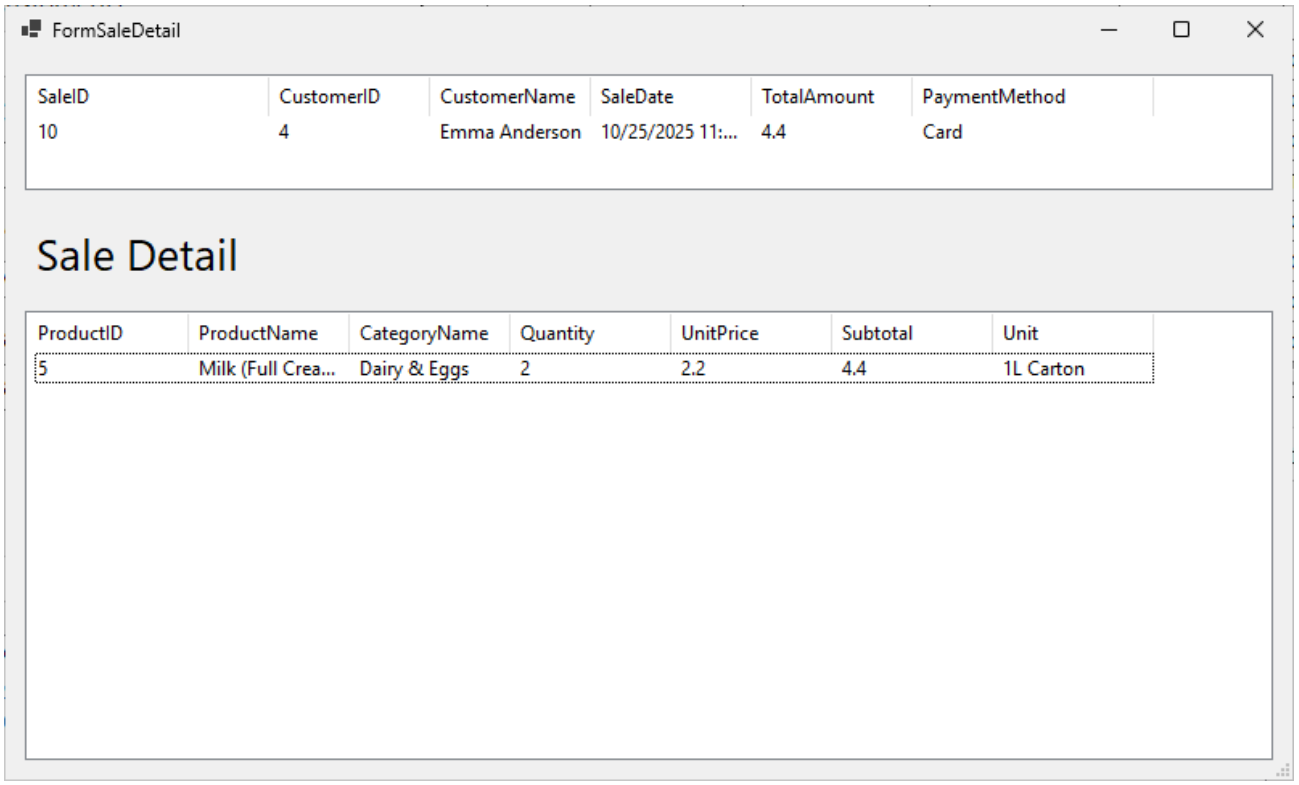
គ. ប៊ូតុងប្រតិបត្តិការផលិតផល:

- Delete Product (លុបផលិតផល)
- Edit Product (កែប្រែផលិតផល)
- Update Product (ធ្វើបច្ចុប្បន្នភាពផលិតផល)
- Add Product (បន្ថែមផលិតផល)

9.4 ប៊ូតុងបញ្ជូនទិន្នន័យចុងក្រោយ (Main Submission Buttons - ផ្នែកខាងក្រោមឆ្វេង)

- Clear (ជម្រះ)
- Update (ធ្វើបច្ចុប្បន្នភាព)
- Submit (ដាក់ស្នើ): បញ្ជូនការលក់ថ្មីទៅក្នុងប្រព័ន្ធ។

9.5 Sale Detail



FormSaleDetail ត្រូវបានហៅចេញពីទម្រង់ការលក់ (FormSale)
ដើម្បីបង្ហាញពីព័ត៌មានលម្អិតនៃទំនិញដែលបានលក់ក្នុងប្រតិបត្តិការនីមួយៗ។

9.5.1 ព័ត៌មានលក់ទូទៅ (Sale Header Information - ផ្នែកខាងលើ)

ផ្នែកនេះបង្ហាញសេចក្តីសង្ខេបអំពីវិក្កយបត្រលក់ដែលបានជ្រើសរើស គឺការលក់លេខ 10 (SaleID 10)៖

- SaleID (លេខសម្គាល់ការលក់): 10
- CustomerID (លេខសម្គាល់អតិថិជន): 4
- CustomerName (ឈ្មោះអតិថិជន): Emma Anderson
- SaleDate (កាលបរិច្ឆេទលក់): 10/25/2025
- TotalAmount (តម្លៃសរុប): 4.4
- PaymentMethod (វិធីទូទាត់): Card (កាត)

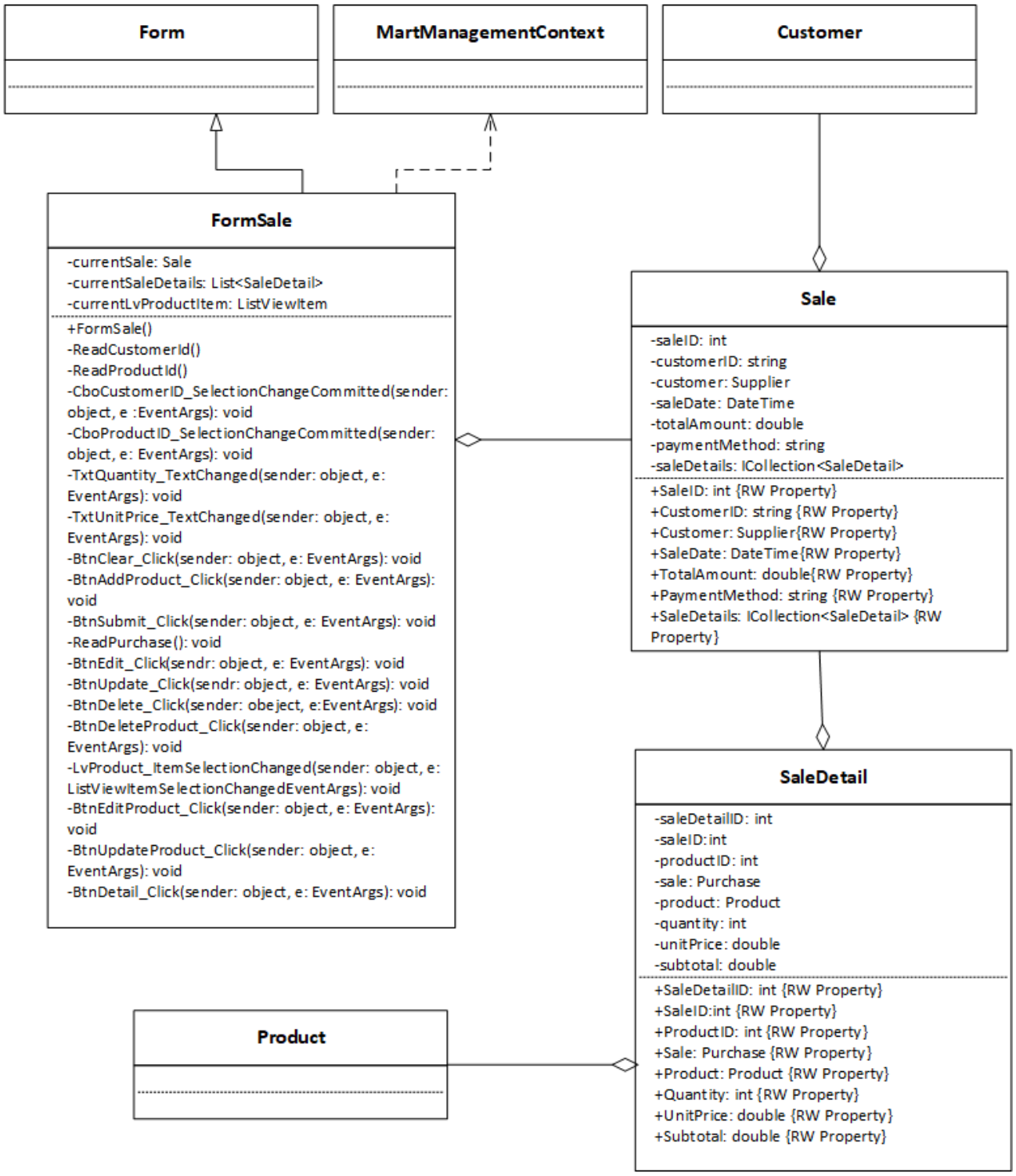
9.5.2 តារាងលម្អិតផលិតផលដែលបានលក់ (Product Detail Table - ផ្នែកកណ្តាល)

នៅក្រោមចំណងជើង "Sale Detail" (លម្អិតនៃការលក់)
មានតារាងមួយដែលបង្ហាញពីផលិតផលដែលបានលក់ចេញក្នុងប្រតិបត្តិការនេះ។ ក្នុងករណីនេះ
មានផលិតផលតែមួយមុខប៉ុណ្ណោះដែលត្រូវបានកត់ត្រា៖

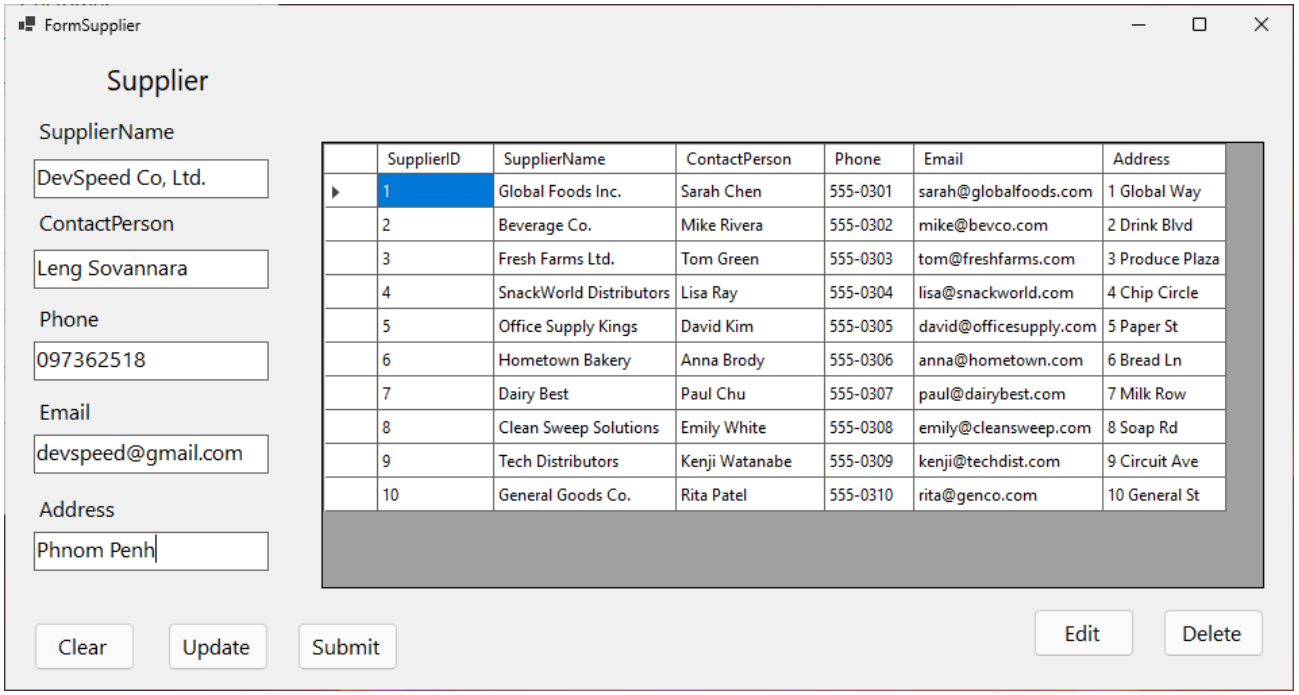
ជួរឈរ (Column)	ព័ត៌មាន (Information)
ProductID (លេខសម្គាល់ផលិតផល)	5
ProductName (ឈ្មោះផលិតផល)	Milk (Full Cream) (ទឹកដោះគោខាប់)
CategoryName (ឈ្មោះប្រភេទ)	Dairy & Eggs (ផលិតផលទឹកដោះគោ និងស៊ុត)
Quantity (បរិមាណ)	2
UnitPrice (តម្លៃលក់ក្នុងមួយឯកតា)	2.2
Subtotal (តម្លៃរង)	4.4
Unit (ឯកតា)	1L Carton (ប្រអប់ចំណុះ 1 លីត្រ)

សរុបមក: ការលក់លេខ 10 ទៅឱ្យអតិថិជន Emma Anderson ដែលមានតម្លៃសរុប 4.4 នេះ
គឺជាការលក់ Milk (Full Cream) ចំនួន 2 ប្រអប់ (តម្លៃលក់ 2.2 ក្នុងមួយប្រអប់)
ដោយទូទាត់តាមរយៈ កាត (Card)។

9.6 UML class diagram



10. Supplier (អ្នកផ្គត់ផ្គង់)



SupplierID	SupplierName	ContactPerson	Phone	Email	Address
1	Global Foods Inc.	Sarah Chen	555-0301	sarah@globalfoods.com	1 Global Way
2	Beverage Co.	Mike Rivera	555-0302	mike@bevco.com	2 Drink Blvd
3	Fresh Farms Ltd.	Tom Green	555-0303	tom@freshfarms.com	3 Produce Plaza
4	SnackWorld Distributors	Lisa Ray	555-0304	lisa@snackworld.com	4 Chip Circle
5	Office Supply Kings	David Kim	555-0305	david@officesupply.com	5 Paper St
6	Hometown Bakery	Anna Brody	555-0306	anna@hometown.com	6 Bread Ln
7	Dairy Best	Paul Chu	555-0307	paul@dairybest.com	7 Milk Row
8	Clean Sweep Solutions	Emily White	555-0308	emily@cleansweep.com	8 Soap Rd
9	Tech Distributors	Kenji Watanabe	555-0309	kenji@techdist.com	9 Circuit Ave
10	General Goods Co.	Rita Patel	555-0310	rita@genco.com	10 General St

FormSupplier អនុញ្ញាតឱ្យអ្នកប្រើប្រាស់មើល កែប្រែ និងបន្ថែមព័ត៌មានក្រុមហ៊ុន ដែលបានលក់ទំនិញឱ្យយើង។

10.1 ផ្នែកបញ្ចូលទិន្នន័យ (Data Input Section - ខាងឆ្វេង)

ទម្រង់នេះមានប្រអប់បញ្ចូលព័ត៌មានលម្អិតអំពីអ្នកផ្គត់ផ្គង់មួយ (Supplier)៖

- SupplierName (ឈ្មោះអ្នកផ្គត់ផ្គង់): បញ្ចូល DevSpeed Co. Ltd.
- ContactPerson (បុគ្គលទំនាក់ទំនង): បញ្ចូល Leng Sovannara
- Phone (ទូរស័ព្ទ): បញ្ចូល 097362518
- Email (អ៊ីមែល): បញ្ចូល devspeed@gmail.com
- Address (អាសយដ្ឋាន): បញ្ចូល Phnom Penh (ភ្នំពេញ)

ប៊ូតុងដែលមាននៅផ្នែកខាងក្រោមនៃការបញ្ចូលទិន្នន័យគឺ៖

- Clear (ជម្រះ): សម្រាប់ជម្រះទិន្នន័យពីប្រអប់ទាំងអស់។
- Update (ធ្វើបច្ចុប្បន្នភាព): សម្រាប់កែប្រែព័ត៌មានអ្នកផ្គត់ផ្គង់ដែលមានស្រាប់។
- Submit (ដាក់ស្នើ): សម្រាប់បញ្ចូលកំណត់ត្រាអ្នកផ្គត់ផ្គង់ថ្មី។

10.2 ផ្នែកតារាងទិន្នន័យ (Data Table Section - ខាងស្តាំ)

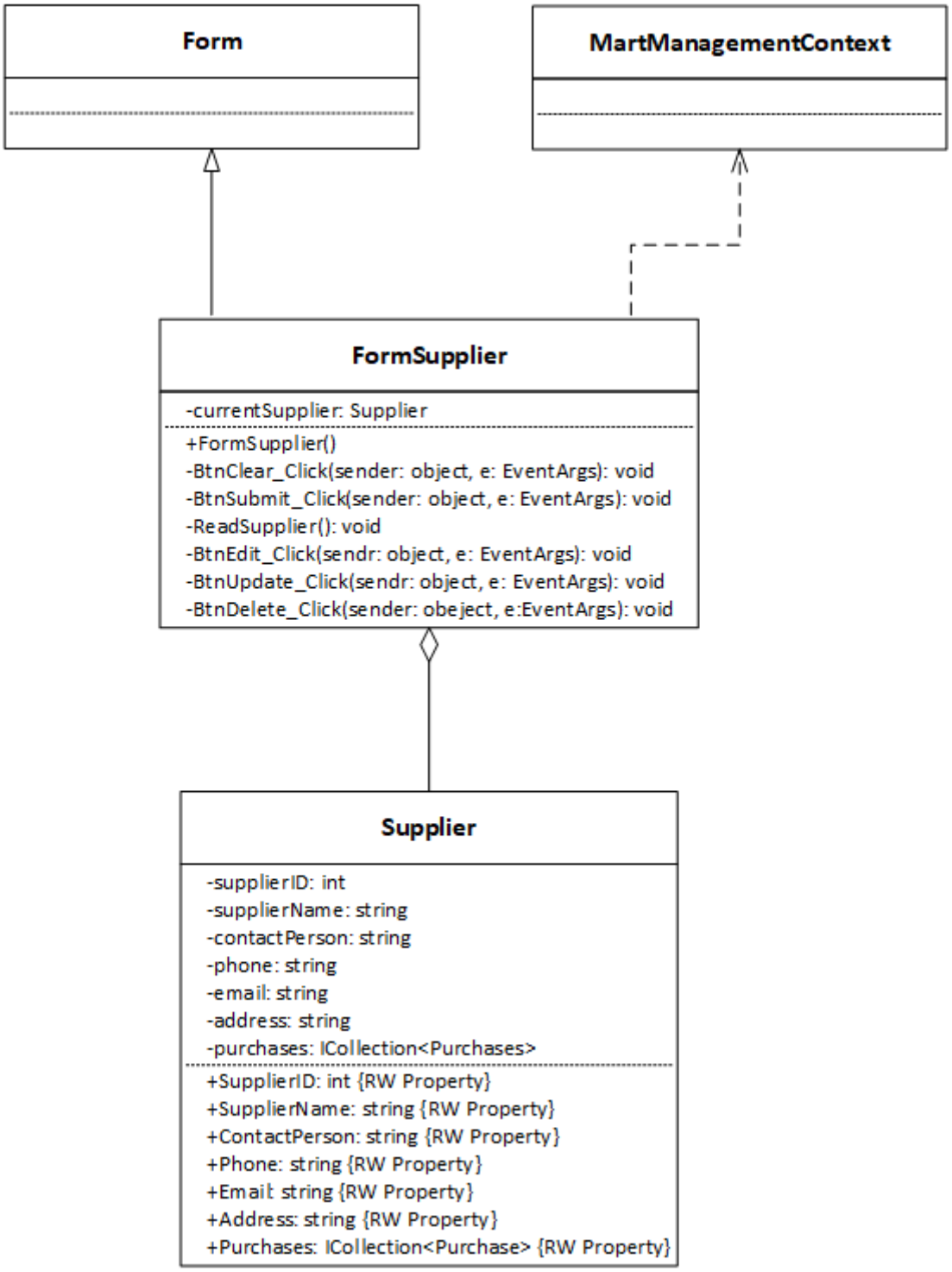
ផ្នែកខាងស្តាំបង្ហាញ តារាងបញ្ជីអ្នកផ្គត់ផ្គង់ ចំនួន ១០ ក្រុមហ៊ុន ដែលមានព័ត៌មានដូចជា៖

- SupplierID (លេខសម្គាល់អ្នកផ្គត់ផ្គង់)
- SupplierName (ឈ្មោះអ្នកផ្គត់ផ្គង់): ដូចជា Global Foods Inc., Beverage Co., Fresh Farms Ltd. ។ល។
- ContactPerson (បុគ្គលទំនាក់ទំនង)
- Phone (ទូរស័ព្ទ)
- Email (អ៊ីមែល)
- Address (អាសយដ្ឋាន)

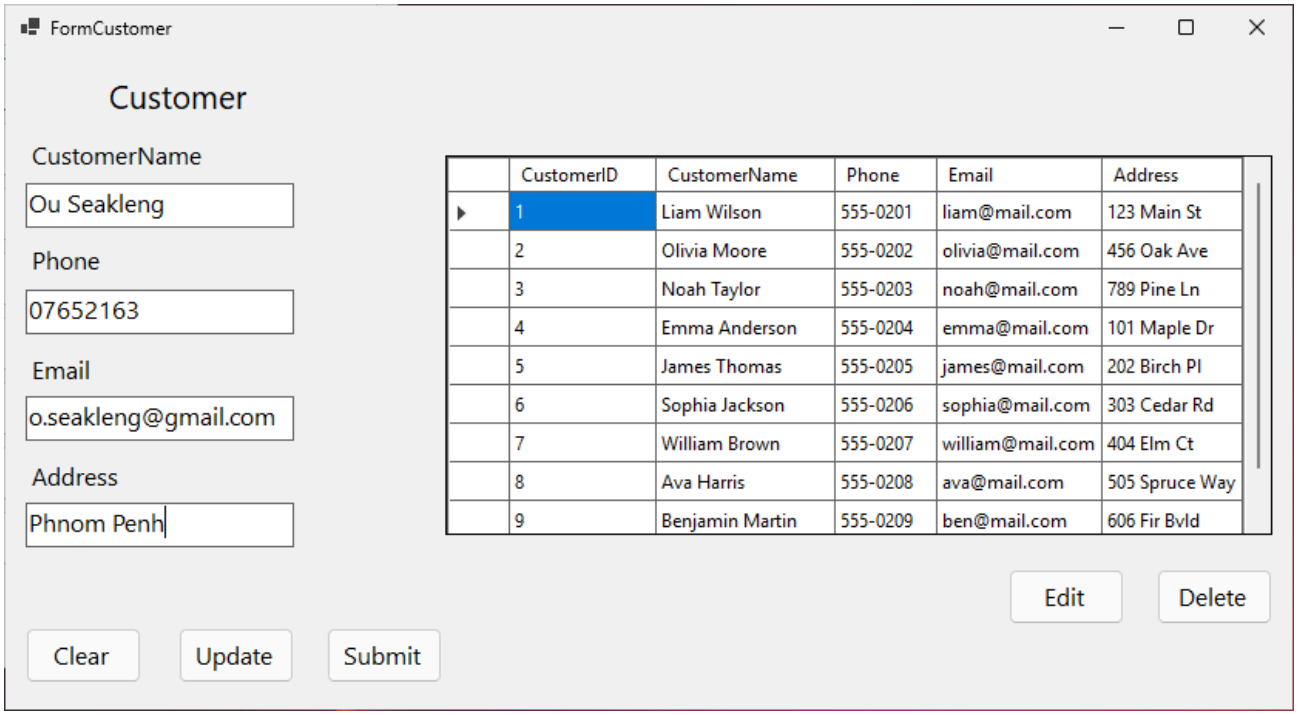
ប៊ូតុងដែលមាននៅក្រោមតារាងគឺ៖

- Edit (កែប្រែ):
  ទំនងជាសម្រាប់ផ្ទុកទិន្នន័យពីជួរដែលបានជ្រើសរើសទៅក្នុងប្រអប់បញ្ចូលខាងឆ្វេង។
- Delete (លុប): សម្រាប់លុបកំណត់ត្រាអ្នកផ្គត់ផ្គង់ដែលបានជ្រើសរើសចេញពីប្រព័ន្ធ។

10.3 UML class diagram



11. Customer (អតិថិជន)



FormCustomer អនុញ្ញាតឱ្យអ្នកប្រើប្រាស់មើល កែប្រែ និងបន្ថែមព័ត៌មានលម្អិតរបស់អតិថិជនទាំងអស់។

11.1 ផ្នែកបញ្ចូលទិន្នន័យ (Data Input Section - ខាងឆ្វេង)

ទម្រង់នេះមានប្រអប់បញ្ចូលព័ត៌មានលម្អិតអំពីអតិថិជនម្នាក់ៗ៖

- CustomerName (ឈ្មោះអតិថិជន): បញ្ចូល Ou Seakleng
- Phone (ទូរស័ព្ទ): បញ្ចូល 07652163
- Email (អ៊ីមែល): បញ្ចូល o.seakleng@gmail.com
- Address (អាសយដ្ឋាន): បញ្ចូល Phnom Penh (ភ្នំពេញ)

ប៊ូតុងដែលមាននៅផ្នែកខាងក្រោមនៃការបញ្ចូលទិន្នន័យគឺ៖

- Clear (ជម្រះ): សម្រាប់ជម្រះទិន្នន័យពីប្រអប់ទាំងអស់។
- Update (ធ្វើបច្ចុប្បន្នភាព): សម្រាប់កែប្រែព័ត៌មានអតិថិជនដែលមានស្រាប់។
- Submit (ដាក់ស្នើ): សម្រាប់បញ្ចូលកំណត់ត្រាអតិថិជនថ្មី។

11.2 ផ្នែកតារាងទិន្នន័យ (Data Table Section - ខាងស្តាំ)

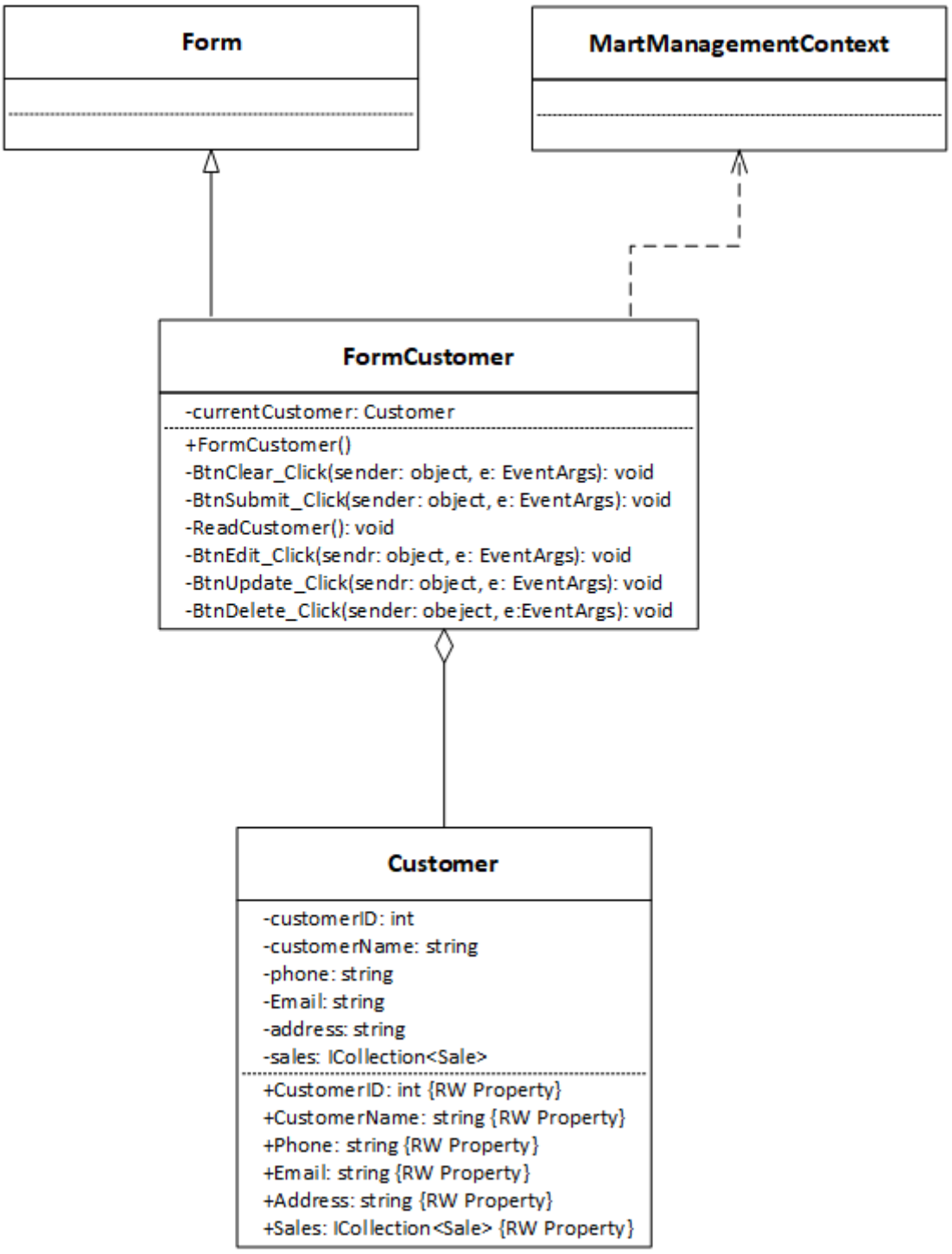
ផ្នែកខាងស្តាំបង្ហាញ តារាងបញ្ជីអតិថិជន ចំនួន ៩ នាក់ ដែលមានព័ត៌មានដូចជា៖

- CustomerID (លេខសម្គាល់អតិថិជន): លេខរៀងពី ១ ដល់ ៩។
- CustomerName (ឈ្មោះអតិថិជន): ដូចជា Liam Wilson, Olivia Moore, Noah Taylor ។ល។
- Phone (ទូរស័ព្ទ)
- Email (អ៊ីមែល)
- Address (អាសយដ្ឋាន)

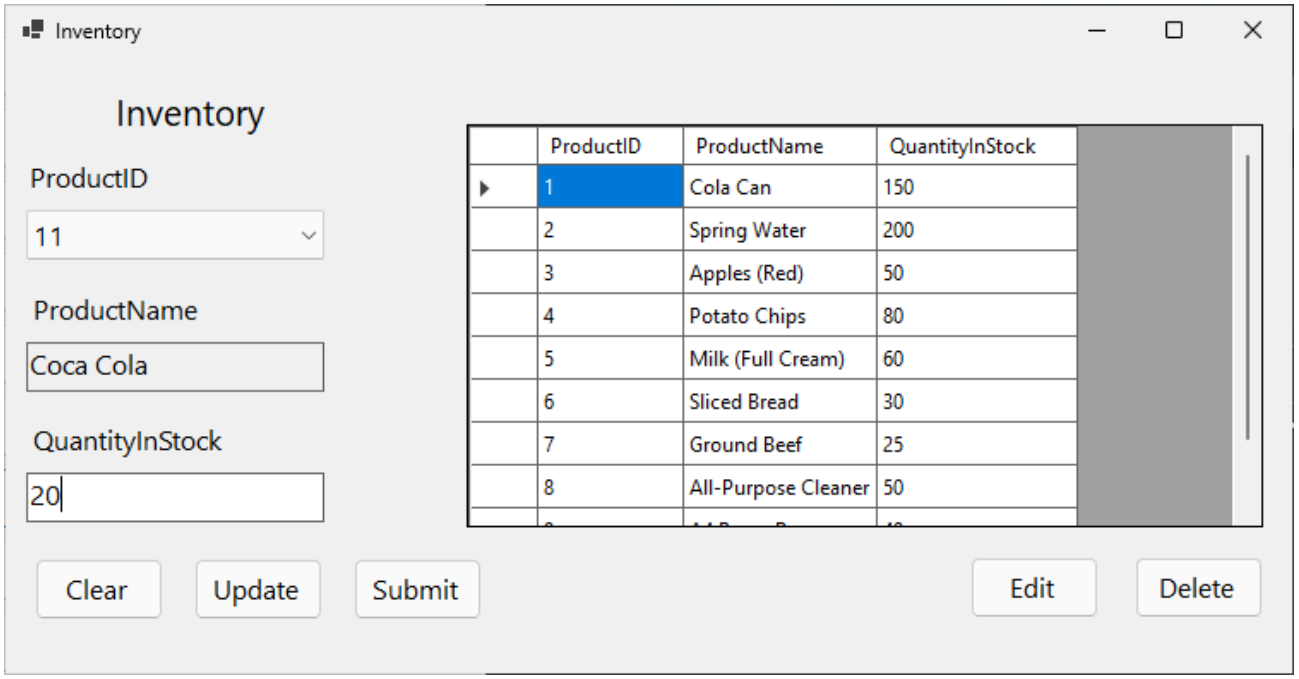
ប៊ូតុងដែលមាននៅក្រោមតារាងគឺ៖

- Edit (កែប្រែ):
  ទំនងជាសម្រាប់ផ្ទុកទិន្នន័យពីជួរដែលបានជ្រើសរើសទៅក្នុងប្រអប់បញ្ចូលខាងឆ្វេង។
- Delete (លុប): សម្រាប់លុបកំណត់ត្រាអតិថិជនដែលបានជ្រើសរើសចេញពីប្រព័ន្ធ។

11.3 UML class diagram



12. Inventory (សារពើភ័ណ្ឌ)



Form Inventory អនុញ្ញាតឱ្យអ្នកប្រើប្រាស់មើល និងធ្វើបច្ចុប្បន្នភាពបរិមាណទំនិញដែលមានក្នុងស្តុក។

12.1 ផ្នែកបញ្ចូលទិន្នន័យ (Data Input Section - ខាងឆ្វេង)

ទម្រង់នេះមានប្រអប់បញ្ចូលព័ត៌មានសម្រាប់គ្រប់គ្រងកម្រិតស្តុកនៃផលិតផលនីមួយៗ៖

- ProductID (លេខសម្គាល់ផលិតផល): បង្ហាញលេខ 11។
- ProductName (ឈ្មោះផលិតផល): បង្ហាញ Coca Cola។
- QuantityInStock (បរិមាណក្នុងស្តុក): បង្ហាញ 20 (ចំនួនដែលមានក្នុងឃ្លាំងបច្ចុប្បន្ន)។

ប៊ូតុងដែលមាននៅផ្នែកខាងក្រោមនៃការបញ្ចូលទិន្នន័យគឺ៖

- Clear (ជម្រះ): សម្រាប់ជម្រះទិន្នន័យពីប្រអប់ទាំងអស់។
- Update (ធ្វើបច្ចុប្បន្នភាព): សម្រាប់កែប្រែបរិមាណស្តុកនៃផលិតផលដែលបានជ្រើសរើស។
- Submit (ដាក់ស្នើ): សម្រាប់បញ្ចូលកំណត់ត្រាស្តុកថ្មី ឬទំនងជាកំណត់ត្រាចាប់ផ្តើម។

12.2 ផ្នែកតារាងទិន្នន័យ (Data Table Section - ខាងស្តាំ)

ផ្នែកខាងស្តាំបង្ហាញ តារាងបញ្ជីស្តុក នៃផលិតផលនានា ដោយមានជួរឈរសំខាន់ៗដូចជា៖

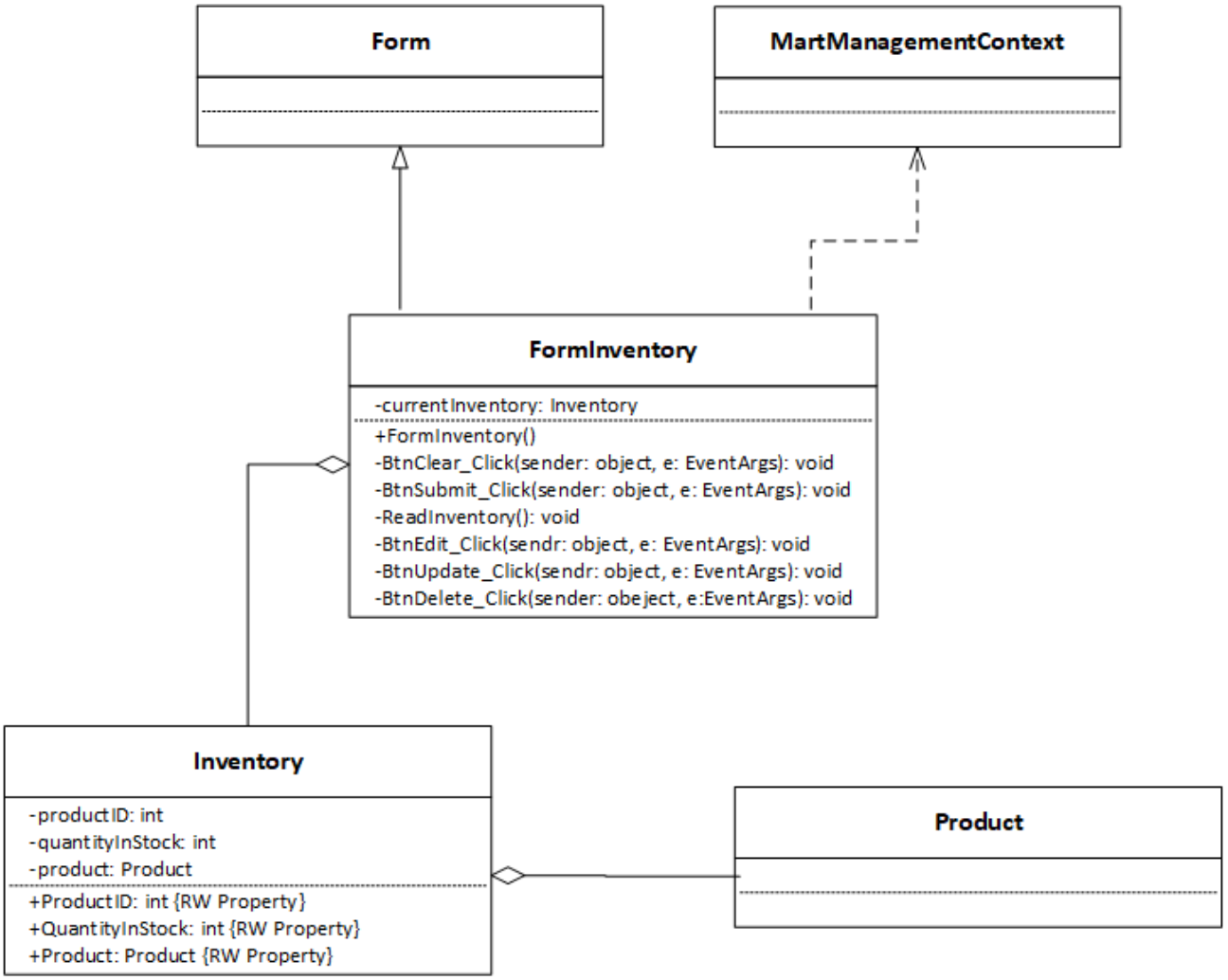
- ProductID (លេខសម្គាល់ផលិតផល): លេខរៀងពី ១ ដល់ ៨ (និងមានទៀតនៅផ្នែកខាងក្រោមដែលត្រូវបានលាក់)។

- ProductName (ឈ្មោះផលិតផល): ដូចជា Cola Can, Spring Water, Apples (Red) ។ល។
- QuantityInStock (បរិមាណក្នុងស្តុក): ចំនួនស្តុកជាក់ស្តែងរបស់ផលិតផលនីមួយៗ (ឧទាហរណ៍ 150, 200, 50, 80, 60, ។ល។)។

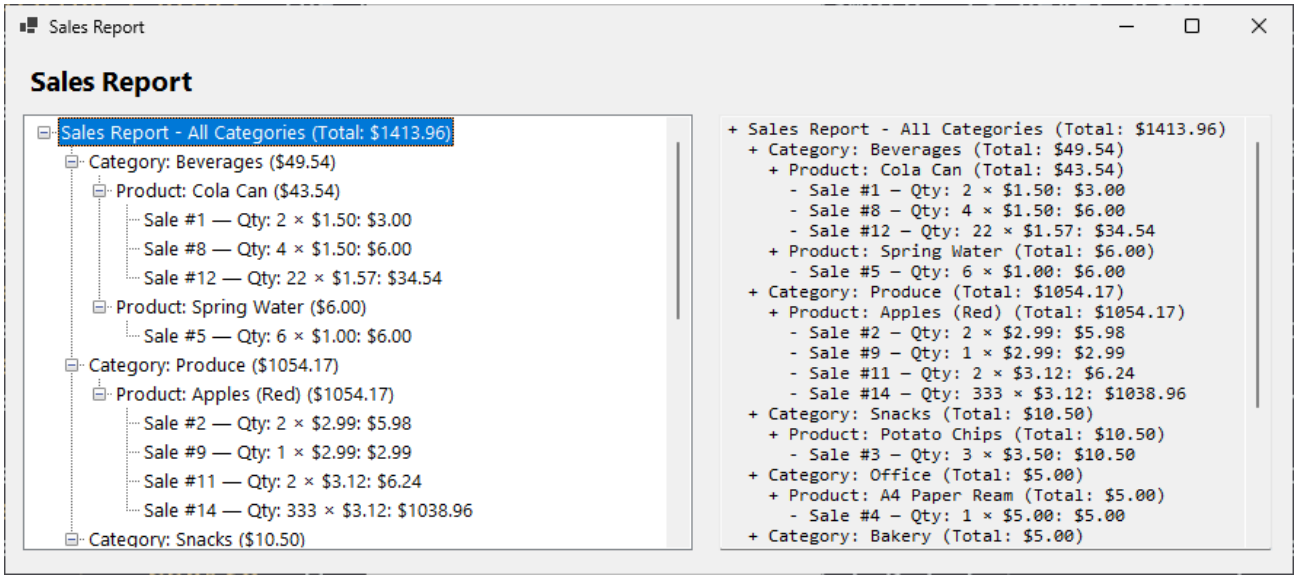
ប៊ូតុងដែលមាននៅក្រោមតារាងគឺ៖

- Edit (កែប្រែ):
  ទំនងជាសម្រាប់ផ្ទុកទិន្នន័យពីជួរដែលបានជ្រើសរើសទៅក្នុងប្រអប់បញ្ចូលខាងឆ្វេង។
- Delete (លុប): សម្រាប់លុបកំណត់ត្រាស្តុកដែលបានជ្រើសរើស។

12.3 UML class diagram



13. Report (របាយការណ៍)



Form Report សម្រាប់បង្ហាញ របាយការណ៍លក់ ដែលមានការបែងចែកជាថ្នាក់ (Hierarchy) យ៉ាងច្បាស់លាស់។ វាត្រូវបានបែងចែកជាពីរផ្នែក៖ ផ្នែកខាងឆ្វេងជាទម្រង់ Tree View និងផ្នែកខាងស្តាំជាទម្រង់ Text View។

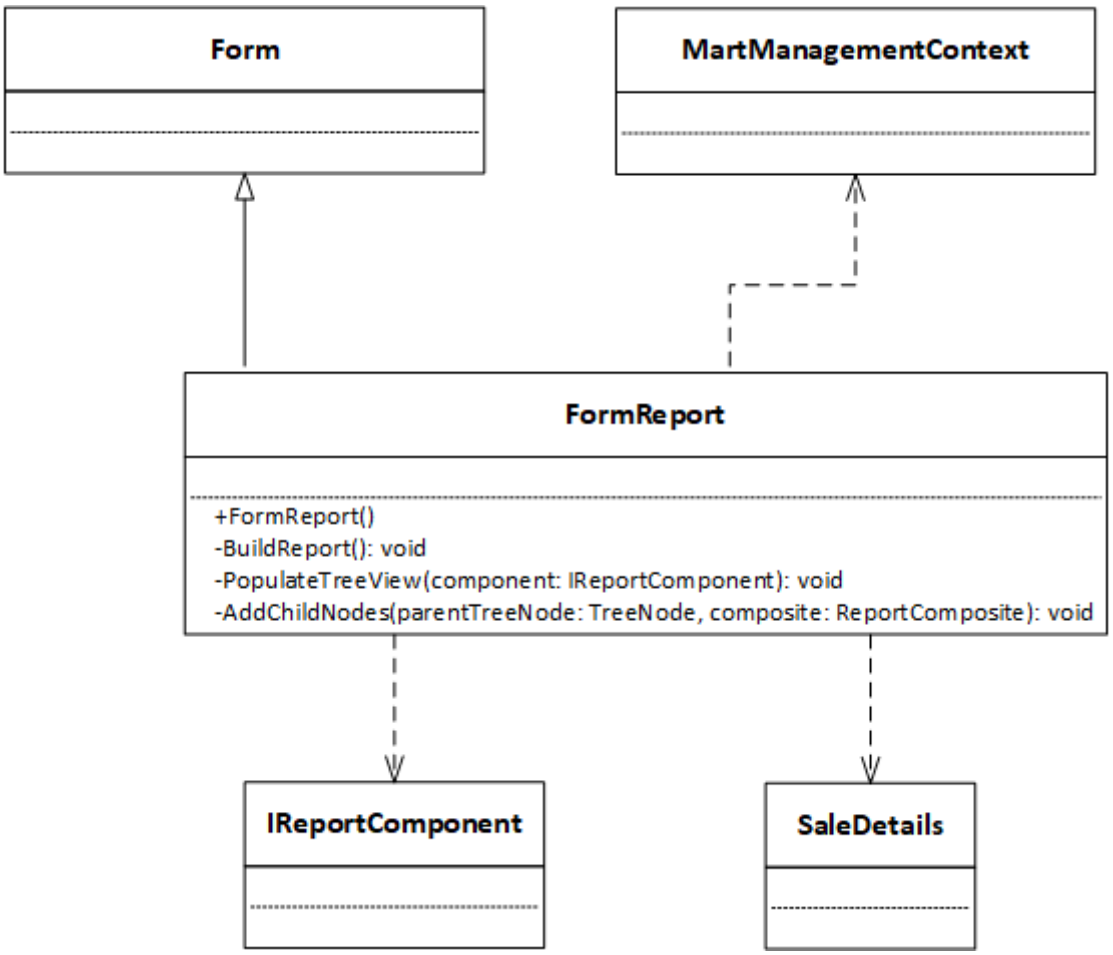
13.1 ព័ត៌មានសំខាន់ៗក្នុងរបាយការណ៍

- កម្រិតខ្ពស់បំផុត៖ បង្ហាញពីសរុបរួមនៃគ្រប់ប្រភេទផលិតផលទាំងអស់ (សរុប៖ $1413.96)។
- ការបែងចែកតាមប្រភេទ (Category): រួមមាន ភេសជ្ជៈ (Beverages), បន្លែផ្លែឈើ (Produce), អាហារសម្រន់ (Snacks), សម្ភារការិយាល័យ (Office) និង នំប៉័ង (Bakery)។
- ព័ត៌មានលម្អិតតាមផលិតផល៖ បង្ហាញឈ្មោះមុខទំនិញ និងចំនួនទឹកប្រាក់សរុបតាមមុខទំនិញនីមួយៗ។
- ព័ត៌មានលម្អិតនៃការលក់ (Sale Details): បង្ហាញពីលេខរៀងប្រតិបត្តិការ ចំនួនលក់ (Qty), តម្លៃក្នុងមួយឯកតា និងតម្លៃសរុបនៃចរន្តលក់នីមួយៗ។

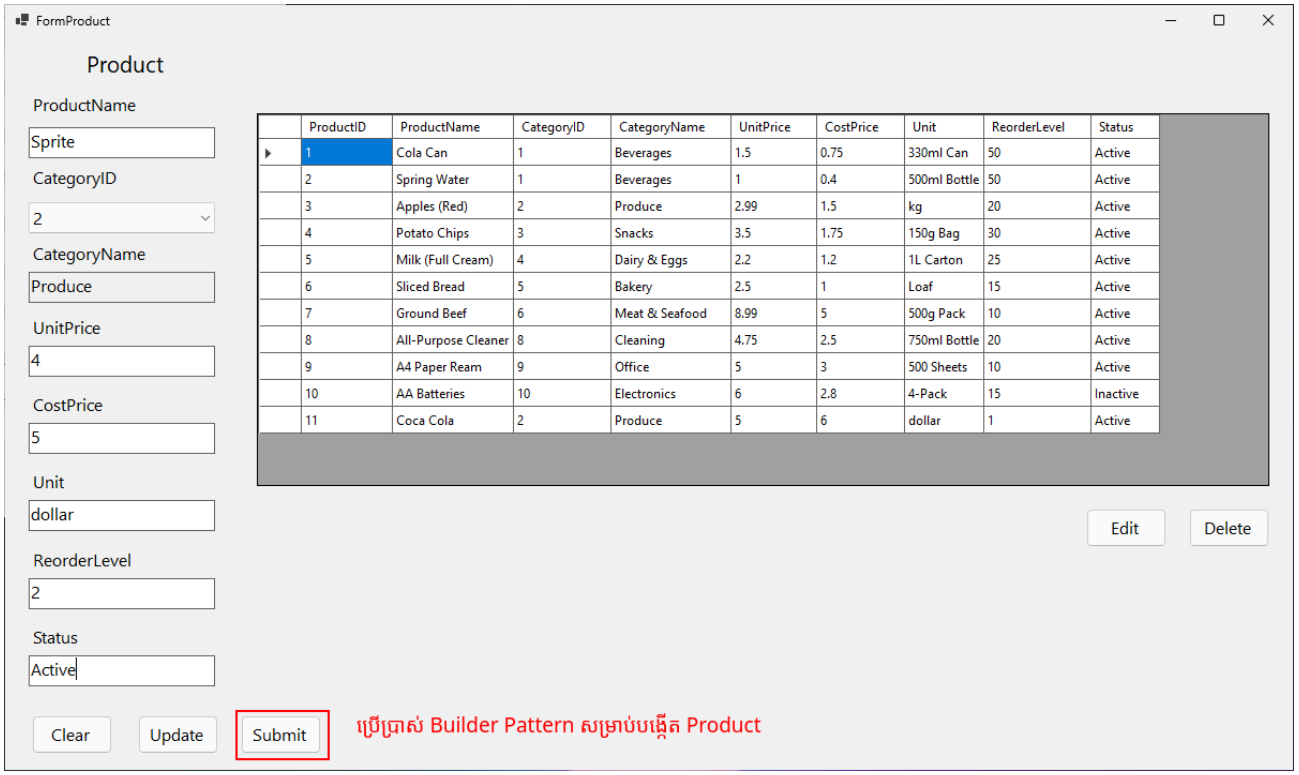
13.2 លក្ខណៈបច្ចេកទេស

- ផ្នែកខាងឆ្វេង៖ ប្រើប្រាស់សញ្ញា + និង - សម្រាប់ពន្លាត ឬបង្រួមមើលព័ត៌មាន។
- ផ្នែកខាងស្តាំ៖ បង្ហាញព័ត៌មានដូចគ្នាជាលក្ខណៈអក្សររៀបតាមលំដាប់លំដោយ ដើម្បីងាយស្រួលអានជារួម។

13.3 UML Class Diagram


Form
MartManagementContext
FormReport
+FormReport()
-BuildReport(): void
-PopulateTreeView(component: IReportComponent): void
-AddChildNodes(parentTreeNode: TreeNode, composite: ReportComposite): void
IReportComponent
SaleDetails

14. Builder Pattern



14.1 ការអនុវត្ត Builder Pattern លើ Form ផលិតផល

Builder Pattern ត្រូវបានប្រើដើម្បីបំបែកដំណើរការនៃការបង្កើត Object ដ៏ស្មុគស្មាញ (Product) ចេញពីការតំណាងរបស់វា។ វាកាត់បន្ថយភាពញ៉ៃវ៉ៃនៃការប្រើ Constructor ដែលមាន Parameter ច្រើនពេក។

14.1.1 ធាតុផ្សំសំខាន់ៗ (Core Components)

- Product Class: ជាថ្នាក់ (Class) ដែលផ្ទុកទិន្នន័យផលិតផលដូចជា `ProductName`, `UnitPrice`, `CostPrice`, `Unit`, `ReorderLevel` និង `Status`។
- ProductBuilder: ជាអ្នកទទួលបន្ទុកកំណត់តម្លៃឱ្យ Attribute នីមួយៗម្តងមួយៗ (Step-by-step) តាមរយៈ Method ដូចជា `.SetName()`, `.SetPrice()`, `.SetStatus()`។
- Submit Button (The Build Method): នៅពេលអ្នកចុចប៊ូតុង Submit វានឹងហៅ Method `.Build()` ដើម្បីប្រមូលរាល់ទិន្នន័យពីគ្រប់ Input Fields មកបង្កើតជា Object តែមួយដែលពេញលេញ។

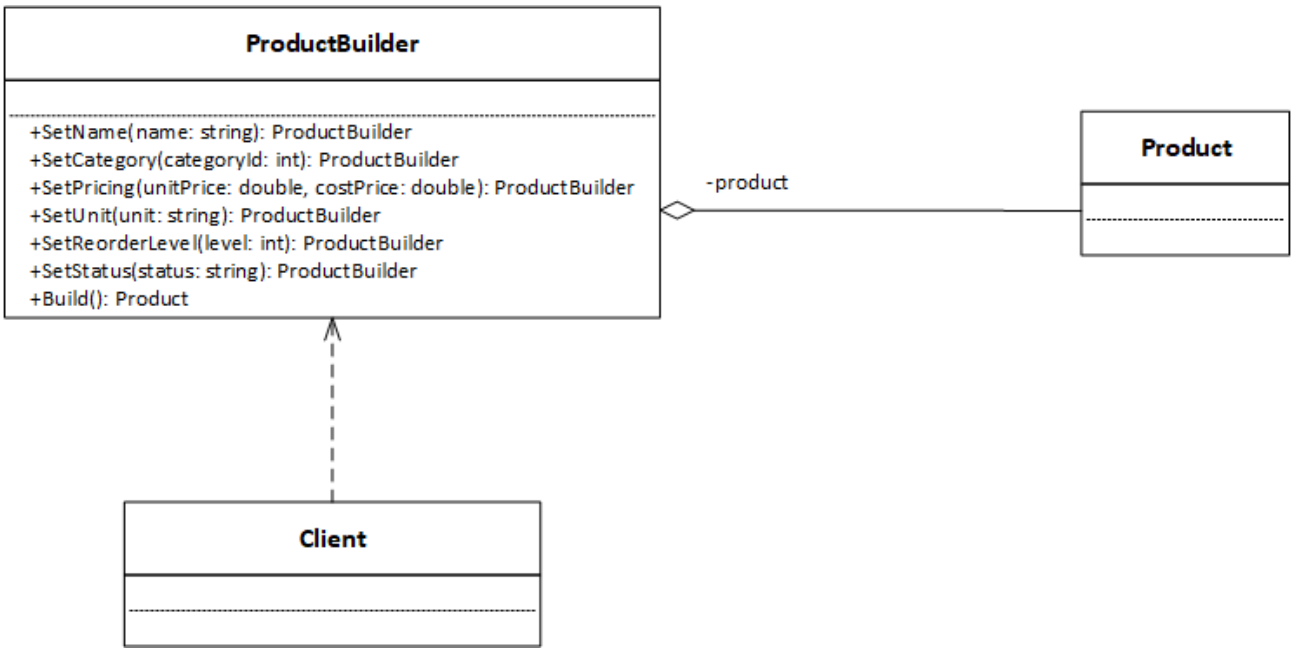
14.1.2. ដំណើរការការងារ (Workflow)

1. Input: អ្នកប្រើប្រាស់បញ្ចូលទិន្នន័យតាមប្រអប់នីមួយៗ (ឧទាហរណ៍៖ "Sprite", "4", "Active")។
2. Assembly: Builder នឹងចាប់យកតម្លៃទាំងនោះមកទុកក្នុងបណ្តោះអាសន្ន។
3. Completion: នៅពេលចុច Submit វានឹងបង្កើត Object ផលិតផលថ្មីមួយ រួចបង្ហាញវានៅក្នុងតារាង (Data Grid) ខាងស្តាំ។

គុណសម្បត្តិក្នុងការប្រើ Pattern នេះ៖

- ភាពច្បាស់លាស់: វាមិនបង្ខំឱ្យយើងបញ្ចូលទិន្នន័យទាំងអស់ក្នុងពេលតែមួយ (Constructor តែមួយ) នោះទេ។
- ភាពងាយស្រួល: បើទោះបីជាផលិតផលខ្លះមិនមាន `ReorderLevel` ឬ `Unit` ក៏ Builder នៅតែអាចបង្កើត Object បានដោយមិនមានកំហុស (Error)។
- ការគ្រប់គ្រង: ងាយស្រួលក្នុងការធ្វើ Validation (ត្រួតពិនិត្យទិន្នន័យ) មុននឹងបង្កើត Object ពិតប្រាកដ។

14.2 UML Class Diagram



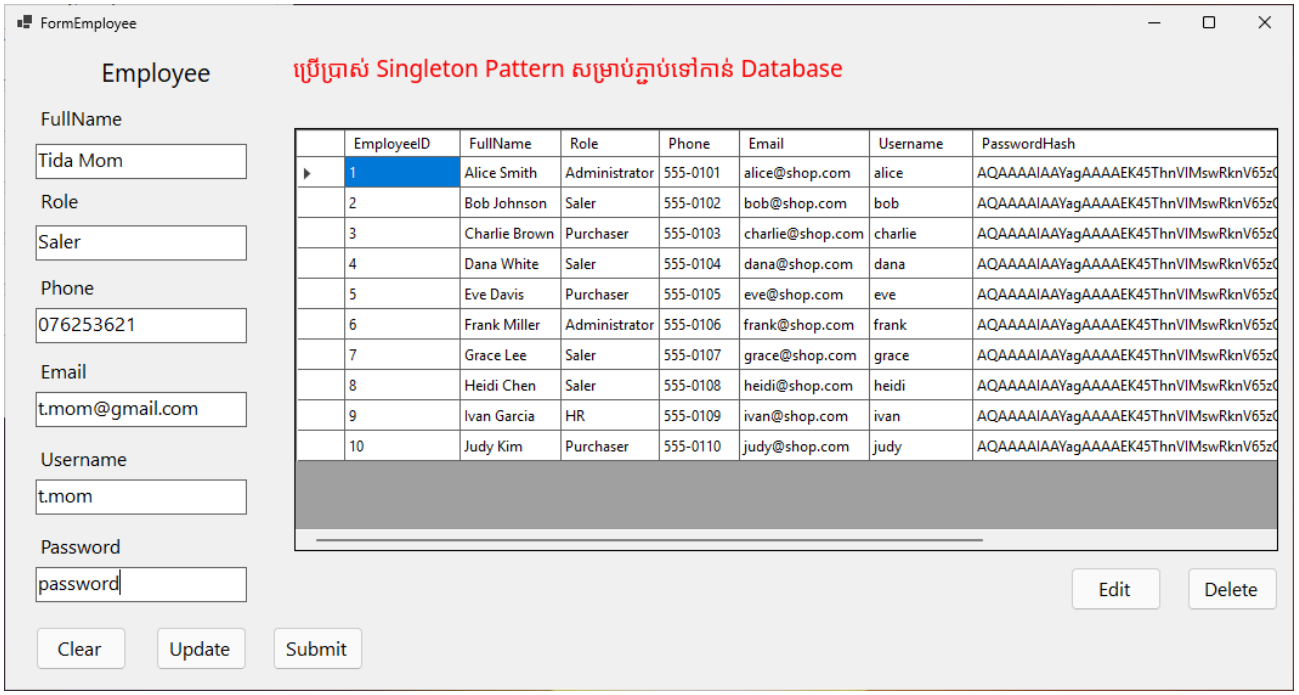
14.3 Implementation Code

```
namespace mart_management
{
    /// <summary>
    /// Builder Pattern: Provides a fluent API for step-by-step
```

```
    /// construction of Product objects.
    /// </summary>
    public class ProductBuilder
    {
        private readonly Product _product = new Product();

        public ProductBuilder SetName(string name)
        {
            _product.ProductName = name;
            return this;
        }

        public ProductBuilder SetCategory(int categoryId)
        {
            _product.CategoryID = categoryId;
            return this;
        }

        public ProductBuilder SetPricing(double unitPrice, double
costPrice)
        {
            _product.UnitPrice = unitPrice;
            _product.CostPrice = costPrice;
            return this;
        }

        public ProductBuilder SetUnit(string unit)
        {
            _product.Unit = unit;
            return this;
        }

        public ProductBuilder SetReorderLevel(int level)
        {
            _product.ReorderLevel = level;
            return this;
        }

        public ProductBuilder SetStatus(string status)
        {
            _product.Status = status;
            return this;
        }
```

```
        /// <summary>
        /// Builds and returns the fully constructed Product.
        /// </summary>
        public Product Build()
        {
            return _product;
        }
    }
}
```

15. Singleton Pattern



15.1 ការអនុវត្ត Singleton Pattern សម្រាប់ការភ្ជាប់ទៅកាន់ Database

Singleton Pattern ត្រូវបានប្រើប្រាស់ដើម្បីគ្រប់គ្រងការភ្ជាប់ (Connection) ទៅកាន់ Database ឱ្យមានតែ មួយគត់ (Single Instance) ក្នុងកម្មវិធីទាំងមូល។

15.1.1 គោលបំណងសំខាន់ (Main Purpose)

- ការសន្សំសំចៃធនធាន (Resource Efficiency): រាល់ពេលដែលអ្នកចុចប៊ូតុង Submit, Update, ឬ Delete កម្មវិធីមិនចាំបាច់បង្កើតការភ្ជាប់ថ្មីទៅកាន់ Database រហូតនោះទេ គឺវាប្រើប្រាស់ការភ្ជាប់ដែលមានស្រាប់។
- ការគ្រប់គ្រងទិន្នន័យចំណុចតែមួយ (Centralized Control):
  រាល់ប្រតិបត្តិការទិន្នន័យបុគ្គលិកទាំងអស់ (ដូចជាការបង្ហាញឈ្មោះក្នុងតារាង)
  គឺហូរតាមច្រកតែមួយ ដើម្បីជៀសវាងការជាន់គ្នានៃទិន្នន័យ។

15.1.2 របៀបដែលវាដំណើរការក្នុង Form នេះ

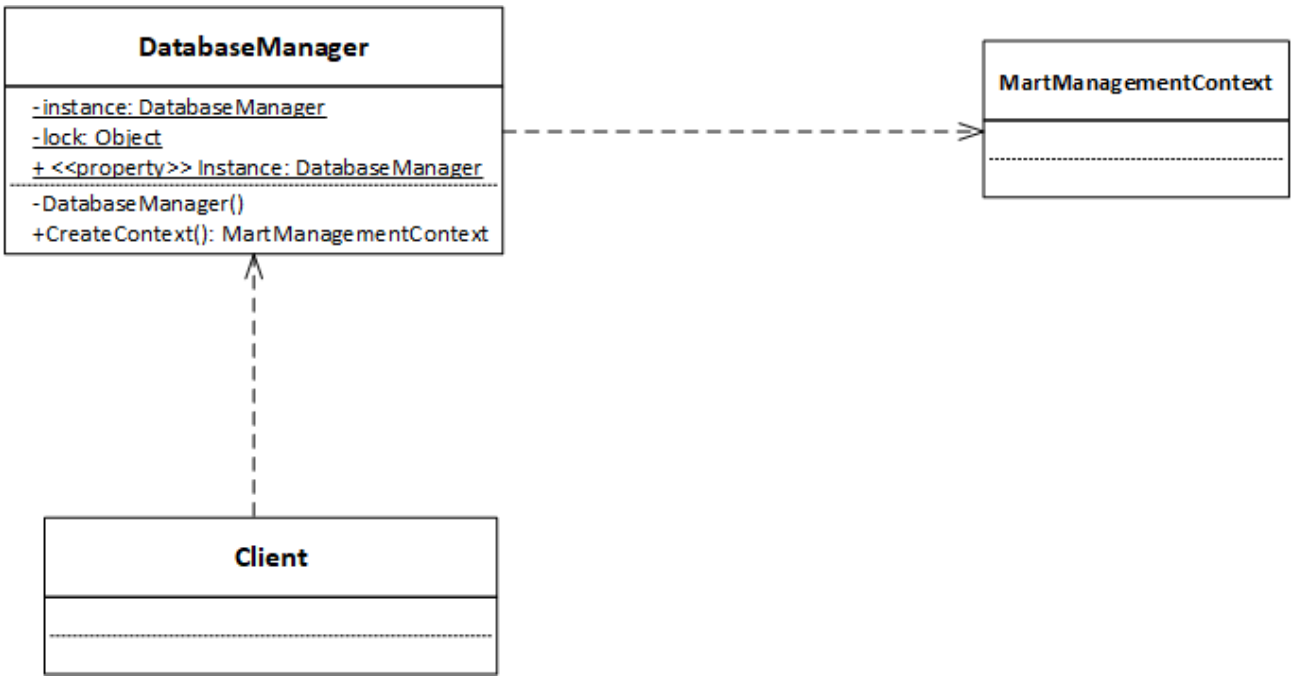
- DatabaseManager Class: មាន Class មួយដែលត្រូវបានរចនាឡើងមិនឱ្យបង្កើត Object ថ្មីតាមរយៈពាក្យគន្លឹះ new បានច្រើនដងឡើយ។

- Global Access: នៅពេល FormEmployee ត្រូវការទាញយកទិន្នន័យមកបង្ហាញក្នុងតារាង (Grid) វានឹងហៅ `DatabaseManager.Instance` ដើម្បីទទួលបានការភ្ជាប់ដែលមានស្រាប់។
- Consistency: ទោះបីជាអ្នកបើក Form បុគ្គលិកនេះច្រើនដងក៏ដោយ ក៏វាប្រើប្រាស់ Database Instance តែមួយដដែល។

គុណសម្បត្តិក្នុងការប្រើ Pattern នេះ៖

- ល្បឿន: កម្មវិធីដើរលឿនជាងមុន ដោយសារមិនចំណាយពេលបង្កើត Connection ទៅ Database ច្រើនដង។
- សុវត្ថិភាព: ធានាថាមានតែ Instance មួយគត់ដែលកាន់កាប់ការសរសេរ ឬកែប្រែទិន្នន័យបុគ្គលិកក្នុងពេលតែមួយ។

15.2 UML Class Diagram



15.3 Implementation Code

```
namespace mart_management
{
    /// <summary>
    /// Singleton Pattern: Ensures a single DatabaseManager instance
    /// controls all DbContext creation throughout the application.
    /// </summary>
```

```
    public sealed class DatabaseManager
    {
        private static DatabaseManager? _instance;
        private static readonly object _lock = new object();

        // Private constructor prevents external instantiation
        private DatabaseManager() { }

        /// <summary>
        /// Thread-safe singleton accessor using double-checked
locking.
        /// </summary>
        public static DatabaseManager Instance
        {
            get
            {
                if (_instance == null)
                {
                    lock (_lock)
                    {
                        if (_instance == null)
                        {
                            _instance = new DatabaseManager();
                        }
                    }
                }
                return _instance;
            }
        }

        /// <summary>
        /// Creates a new MartManagementContext instance.
        /// All database access should go through this method.
        /// </summary>
        public MartManagementContext CreateContext()
        {
            return new MartManagementContext();
        }
    }
}
```

16. Composite Pattern

16.1 ការអនុវត្ត Composite Pattern លើ Sales Report

Composite Pattern ត្រូវបានប្រើដើម្បីរៀបចំទិន្នន័យដែលមានលក្ខណៈជា Tree Structure (ហានានុក្រម) ដែលអនុញ្ញាតឱ្យយើងគ្រប់គ្រង "វត្ថុទោល" (Individual objects) និង "ក្រុមនៃវត្ថុ" (Composites) តាមរបៀបដូចគ្នា។

16.1.1 រចនាសម្ព័ន្ធហានានុក្រម (Hierarchy Structure)

នៅក្នុង Form នេះ ទិន្នន័យត្រូវបានបែងចែកជាថ្នាក់ៗដូចខាងក្រោម៖

- Root (មេធំ): Sales Report - All Categories (ជារបាយការណ៍សរុបរួម)។
- Composite (ក្រុម): Category (ឧទាហរណ៍៖ Beverages, Produce) ដែលផ្ទុកទៅដោយផលិតផលជាច្រើន។
- Sub-Composite: Product (ឧទាហរណ៍៖ Cola Can, Apples) ដែលផ្ទុកទៅដោយប្រតិបត្តិការលក់ (Sales) បន្តទៀត។
- Leaf (ស្លឹក/ថ្នាំងចុងក្រោយ): ការលក់នីមួយៗ (Sale #1, Sale #8) ដែលជាទិន្នន័យតូចបំផុត។

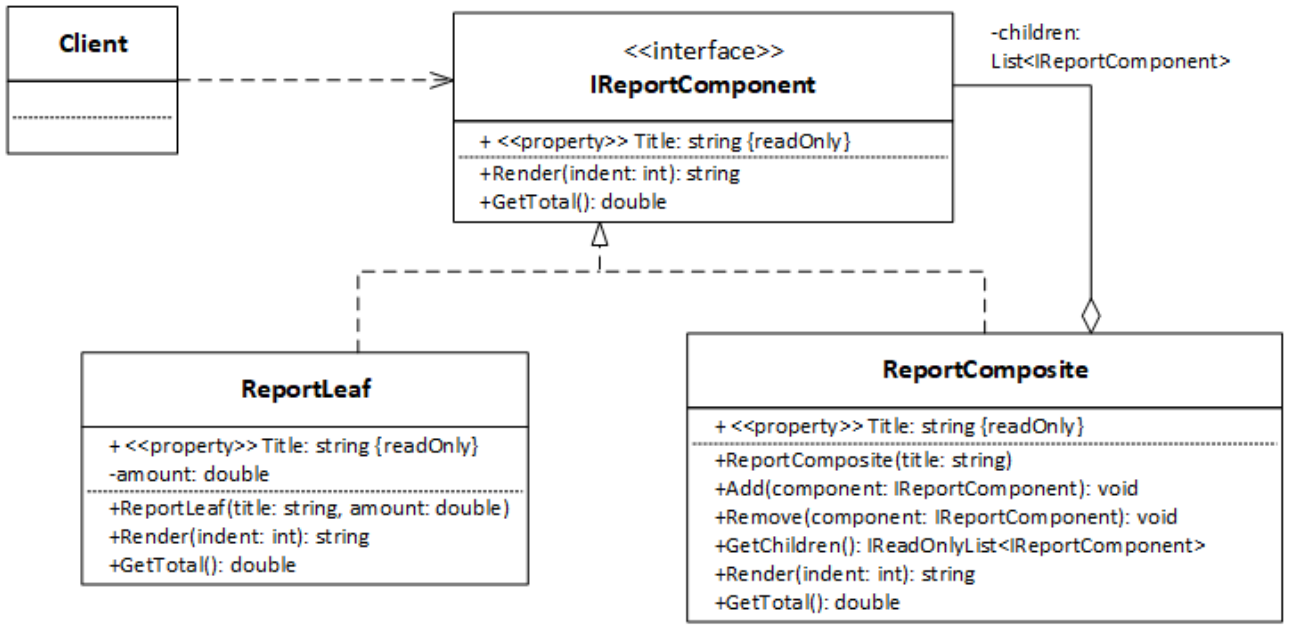
16.1.2 របៀបដែលវាដំណើរការក្នុង Form នេះ

- ការគណនាសរុប (Recursive Aggregation): នៅពេលមេធំចង់ដឹងតម្លៃសរុប ($1413.96) វានឹងសួរទៅកាន់ Category នីមួយៗ ហើយ Category នីមួយៗនឹងសួរទៅកាន់ Product រួចបូកសរុបឡើងមកវិញដោយស្វ័យប្រវត្តិ។
- ការបង្ហាញលទ្ធផល (Uniform Display): មិនថាជា Category ឬជា Product នោះទេ វានឹងបង្ហាញឈ្មោះ និងតម្លៃក្នុងទម្រង់ស្រដៀងគ្នា ដែលធ្វើឱ្យ TreeView ងាយស្រួលរៀបចំ។

គុណសម្បត្តិក្នុងការប្រើ Pattern នេះ៖

- ភាពបត់បែន: អ្នកអាចបន្ថែម Category ថ្មី ឬ Product ថ្មីទៅក្នុង Tree បានយ៉ាងងាយស្រួល ដោយមិនប៉ះពាល់ដល់រចនាសម្ព័ន្ធកូដរួម។
- ភាពងាយស្រួល: កូដដែលប្រើសម្រាប់គណនាតម្លៃ ឬបង្ហាញឈ្មោះ អាចប្រើរួមគ្នាបានទាំងលើវត្ថុទោល និងវត្ថុជាក្រុម។

16.2 UML Class Diagram



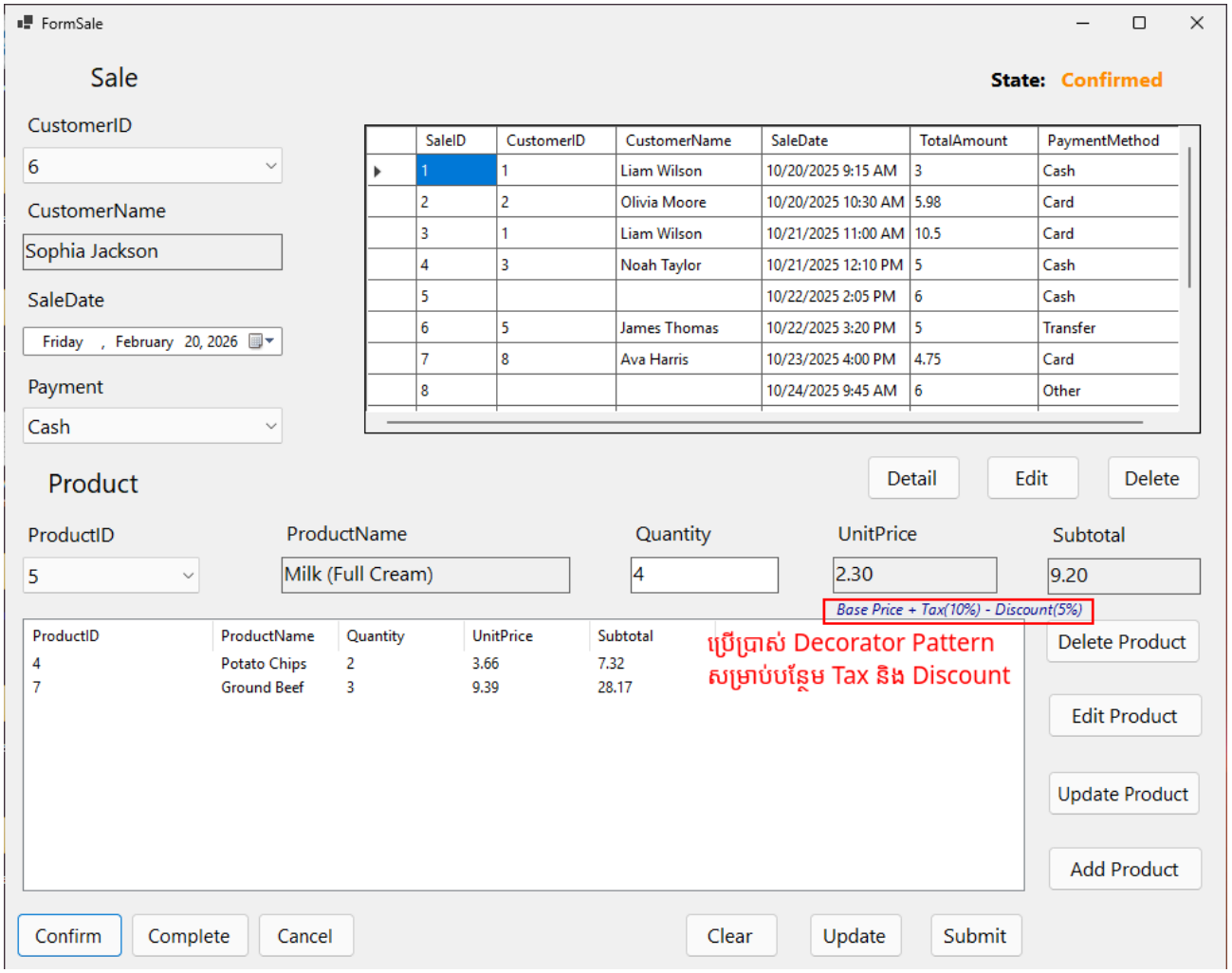
16.3 Implementation Code

```
namespace mart_management
{
    /// <summary>
    /// Composite Pattern: Uniform interface for both individual
    /// report items (leaf) and groups of items (composite).
    /// </summary>
    public interface IReportComponent
```

```
    {
        string Title { get; }
        string Render(int indent = 0);
        double GetTotal();
    }

    /// <summary>
    /// Leaf node: Represents a single report line item (e.g., one sale
detail).
    /// </summary>
    public class ReportLeaf : IReportComponent
    {
        public string Title { get; }
        private readonly double _amount;

        public ReportLeaf(string title, double amount)
        {
            Title = title;
            _amount = amount;
        }

        public string Render(int indent = 0)
        {
            string pad = new string(' ', indent * 2);
            return $"{pad}- {Title}: ${_amount:F2}";
        }

        public double GetTotal() => _amount;
    }

    /// <summary>
    /// Composite node: Contains child components (both leaves and
other composites).
    /// Represents a group like "Category" or "Product" in the report
hierarchy.
    /// </summary>
    public class ReportComposite : IReportComponent
    {
        public string Title { get; }
        private readonly List<IReportComponent> _children = new
List<IReportComponent>();

        public ReportComposite(string title)
        {
```

```
            Title = title;
        }

        public void Add(IReportComponent component)
        {
            _children.Add(component);
        }

        public void Remove(IReportComponent component)
        {
            _children.Remove(component);
        }

        public IReadOnlyList<IReportComponent> GetChildren() =>
_children.AsReadOnly();

        public string Render(int indent = 0)
        {
            string pad = new string(' ', indent * 2);
            var sb = new System.Text.StringBuilder();
            sb.AppendLine($"{pad}+ {Title} (Total: ${GetTotal():F2})");
            foreach (var child in _children)
            {
                sb.AppendLine(child.Render(indent + 1));
            }
            return sb.ToString().TrimEnd();
        }

        public double GetTotal()
        {
            double total = 0;
            foreach (var child in _children)
            {
                total += child.GetTotal();
            }
            return total;
        }
    }
}
```

17. Decorator Pattern



17.1 ការអនុវត្ត Decorator Pattern សម្រាប់ Tax និង Discount

Decorator Pattern ត្រូវបានប្រើដើម្បីបន្ថែមមុខងារ ឬការទទួលខុសត្រូវថ្មីៗទៅឱ្យ Object មួយ (ក្នុងករណីនេះគឺតម្លៃទំនិញ) ដោយមិនចាំបាច់កែប្រែរចនាសម្ព័ន្ធដើមរបស់វា។

17.1.1 គោលបំណងសំខាន់ (Main Purpose)

- ការបន្ថែមលក្ខណៈពិសេស (Flexible Wrappers): វាអនុញ្ញាតឱ្យយើងយក "តម្លៃដើម" (Base Price) មកស្រោបបន្ថែមដោយ "ពន្ធ" (Tax) ឬ "ការបញ្ចុះតម្លៃ" (Discount) តាមតម្រូវការជាក់ស្តែង។
- ជៀសវាងការបង្កើត Class ច្រើនពេក: យើងមិនចាំបាច់បង្កើត Class ផ្សេងៗគ្នាសម្រាប់ "តម្លៃដែលមានពន្ធ" ឬ "តម្លៃដែលមានបញ្ចុះតម្លៃ" ឡើយ គឺយើងគ្រាន់តែប្រើ Decorator ដើម្បីបន្ថែមពួកវានៅពេល Runtime។

17.1.2 របៀបដែលវាដំណើរការក្នុង Form នេះ

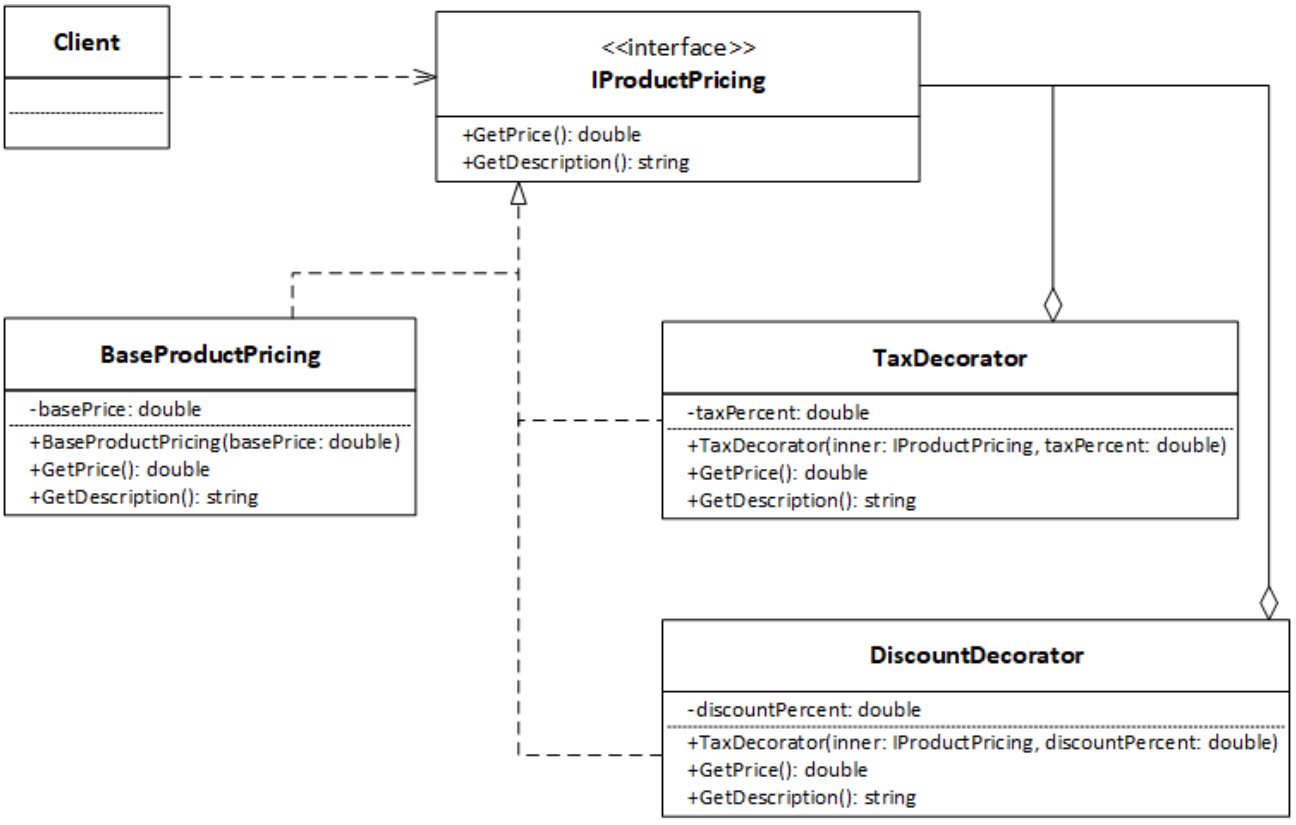
យោងតាមចំណងជើងពណ៌ក្រហមក្នុងរូបភាព៖ "Base Price + Tax(10%) - Discount(5%)"

- Component: គឺជាតម្លៃទំនិញដើម (Base Price)។
- Decorator 1 (Tax): ធ្វើការស្រោបពីលើតម្លៃដើម ដើម្បីគណនាបន្ថែមពន្ធ 10%។
- Decorator 2 (Discount): ធ្វើការស្រោបបន្តទៀត ដើម្បីដកតម្លៃបញ្ចុះ 5% ចេញពីលទ្ធផលចុងក្រោយ។

គុណសម្បត្តិក្នុងការប្រើ Pattern នេះ៖

- ភាពងាយស្រួលក្នុងការផ្សំគ្នា (Combinable): អ្នកអាចជ្រើសរើសបន្ថែមតែ Tax, តែ Discount ឬថែមទាំងពីរក្នុងពេលតែមួយបានយ៉ាងបត់បែន។
- Clean Code: កូដសម្រាប់គណនាពន្ធ និងការបញ្ចុះតម្លៃត្រូវបានបំបែកដាច់ដោយឡែកពីគ្នា ដែលធ្វើឱ្យងាយស្រួលក្នុងការកែប្រែភាគរយនាពេលអនាគត។

17.2 UML Class Diagram

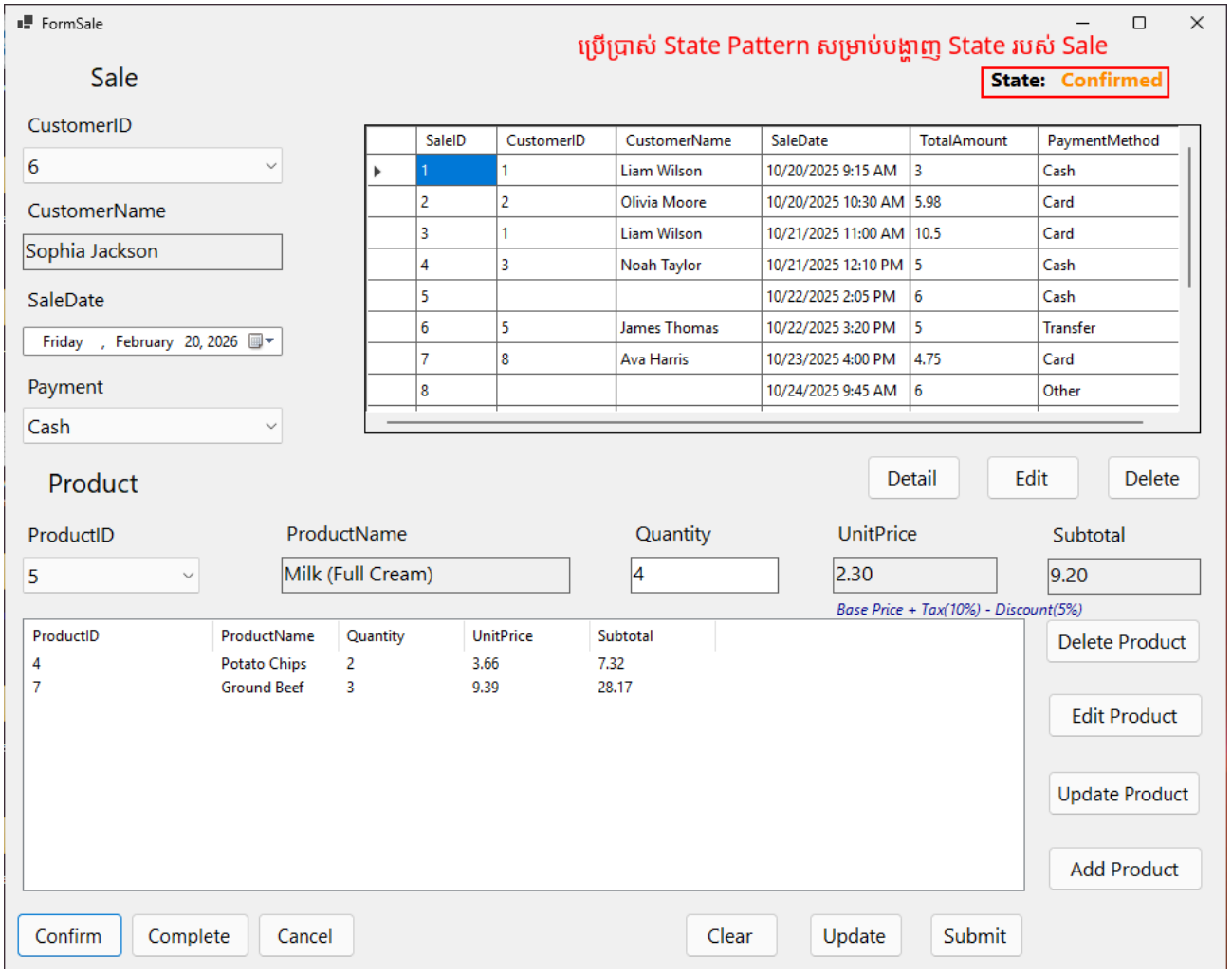


17.3 Implementation Code

```
namespace mart_management
{
    /// <summary>
    /// Decorator Pattern: Interface for product pricing that
    /// can be wrapped with additional behavior.
    /// </summary>
    public interface IProductPricing
    {
        double GetPrice();
        string GetDescription();
    }

    /// <summary>
    /// Base (concrete) implementation: returns the raw product unit
price.
    /// </summary>
    public class BaseProductPricing : IProductPricing
    {
        private readonly double _basePrice;

        public BaseProductPricing(double basePrice)
        {
            _basePrice = basePrice;
        }

        public double GetPrice() => _basePrice;
        public string GetDescription() => "Base Price";
    }

    /// <summary>
    /// Decorator: Adds a tax percentage on top of the wrapped pricing.
    /// </summary>
    public class TaxDecorator : IProductPricing
    {
        private readonly IProductPricing _inner;
        private readonly double _taxPercent;

        public TaxDecorator(IProductPricing inner, double taxPercent)
        {
            _inner = inner;
            _taxPercent = taxPercent;
        }

        public double GetPrice()
```

```
        {
            return _inner.GetPrice() * (1 + _taxPercent / 100.0);
        }

        public string GetDescription()
        {
            return $"{_inner.GetDescription()} + Tax({_taxPercent}%)";
        }
    }

    /// <summary>
    /// Decorator: Applies a discount percentage on top of the wrapped
pricing.
    /// </summary>
    public class DiscountDecorator : IProductPricing
    {
        private readonly IProductPricing _inner;
        private readonly double _discountPercent;

        public DiscountDecorator(IProductPricing inner, double
discountPercent)
        {
            _inner = inner;
            _discountPercent = discountPercent;
        }

        public double GetPrice()
        {
            return _inner.GetPrice() * (1 - _discountPercent / 100.0);
        }

        public string GetDescription()
        {
            return $"{_inner.GetDescription()} -
Discount({_discountPercent}%)";
        }
    }
}
```

18. State Pattern



18.1 ការអនុវត្ត State Pattern សម្រាប់គ្រប់គ្រងស្ថានភាពនៃការលក់ (Sale State)

State Pattern ត្រូវបានប្រើដើម្បីអនុញ្ញាតឱ្យ Object (ក្នុងករណីនេះគឺការលក់ - Sale) ផ្លាស់ប្តូរឥរិយាបថរបស់វា នៅពេលដែលស្ថានភាពផ្ទៃក្នុងរបស់វាផ្លាស់ប្តូរ។

18.1.1 ការផ្លាស់ប្តូរស្ថានភាព (State Transitions)

នៅក្នុង Form នេះ យើងឃើញមានស្ថានភាពខុសៗគ្នានៃការលក់ ដែលត្រូវបានបង្ហាញនៅជ្រុងខាងស្តាំខាងលើ (ឧទាហរណ៍៖ State: Confirmed)៖

- Draft (សេចក្តីព្រាង): ស្ថានភាពដំបូងនៅពេលបញ្ចូលទិន្នន័យ។
- Confirmed (បានបញ្ជាក់): បន្ទាប់ពីចុចប៊ូតុង Confirm។
- Completed (បានបញ្ចប់): បន្ទាប់ពីចុចប៊ូតុង Complete។
- Canceled (បានបោះបង់): បន្ទាប់ពីចុចប៊ូតុង Cancel។

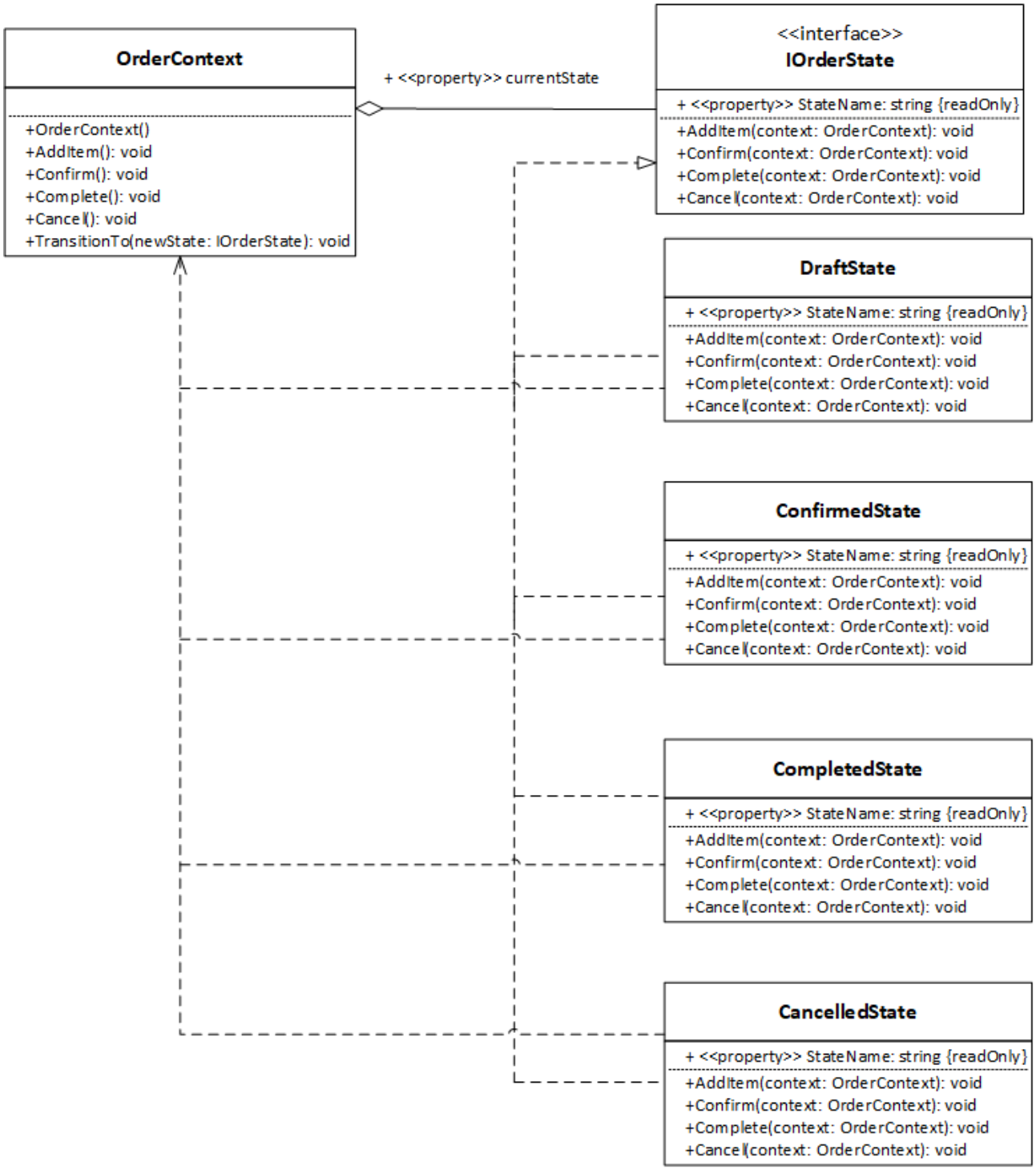
18.1.2 របៀបដែលវាដំណើរការក្នុង Form នេះ៖

- Behavior Change: នៅពេលស្ថានភាពផ្លាស់ប្តូរ ឥរិយាបថរបស់ប៊ូតុង ឬការកែប្រែទិន្នន័យនឹងប្រែប្រួល។ ឧទាហរណ៍៖ ប្រសិនបើស្ថានភាពជា "Confirmed" អ្នកប្រហែលជាមិនអាចកែប្រែ (Edit) ចំនួនទំនិញបានទៀតទេ។
- Context: Form នេះដើរតួជា "Context" ដែលផ្ទុក Object នៃ State បច្ចុប្បន្ន។
- Clean Logic: យើងមិនចាំបាច់ប្រើ `if-else` ឬ `switch-case` ច្រើនជាន់ដើម្បីឆែកស្ថានភាពនោះទេ ព្រោះរាល់ Logic ត្រូវបានបំបែកទៅតាម Class នៃ State នីមួយៗ។

គុណសម្បត្តិក្នុងការប្រើ Pattern នេះ៖

- ភាពងាយស្រួលក្នុងការបន្ថែម State: បើថ្ងៃក្រោយអ្នកចង់ថែម State "Refunded" (ការបង្វិលលុយ) អ្នកគ្រាន់តែបង្កើត Class ថ្មីមួយទៀតដោយមិនប៉ះពាល់ដល់កូដចាស់។
- កាត់បន្ថយភាពស្មុគស្មាញ: ធ្វើឱ្យកូដមានរបៀបរៀបរយ និងងាយស្រួលអាន ព្រោះសកម្មភាពនីមួយៗត្រូវបានកំណត់យ៉ាងច្បាស់លាស់ទៅតាមស្ថានភាពរបស់វា។

18.2 UML Class Diagram



18.3 Implementation Code

```
using System.Windows.Forms;

namespace mart_management
{
    /// <summary>
    /// State Pattern: Interface for sale/order lifecycle states.
```

```
    /// Each state determines what actions are allowed.
    /// </summary>
    public interface IOrderState
    {
        string StateName { get; }
        void AddItem(OrderContext context);
        void Confirm(OrderContext context);
        void Complete(OrderContext context);
        void Cancel(OrderContext context);
    }

    /// <summary>
    /// State machine context: Holds the current state and delegates
actions.
    /// </summary>
    public class OrderContext
    {
        public IOrderState CurrentState { get; set; }

        public OrderContext()
        {
            CurrentState = new DraftState();
        }

        public void AddItem() => CurrentState.AddItem(this);
        public void Confirm() => CurrentState.Confirm(this);
        public void Complete() => CurrentState.Complete(this);
        public void Cancel() => CurrentState.Cancel(this);

        public void TransitionTo(IOrderState newState)
        {
            CurrentState = newState;
        }
    }

    /// <summary>
    /// Draft State: The order is being built. Items can be added.
    /// Can transition to Confirmed or Cancelled.
    /// </summary>
    public class DraftState : IOrderState
    {
        public string StateName => "Draft";

        public void AddItem(OrderContext context)
```

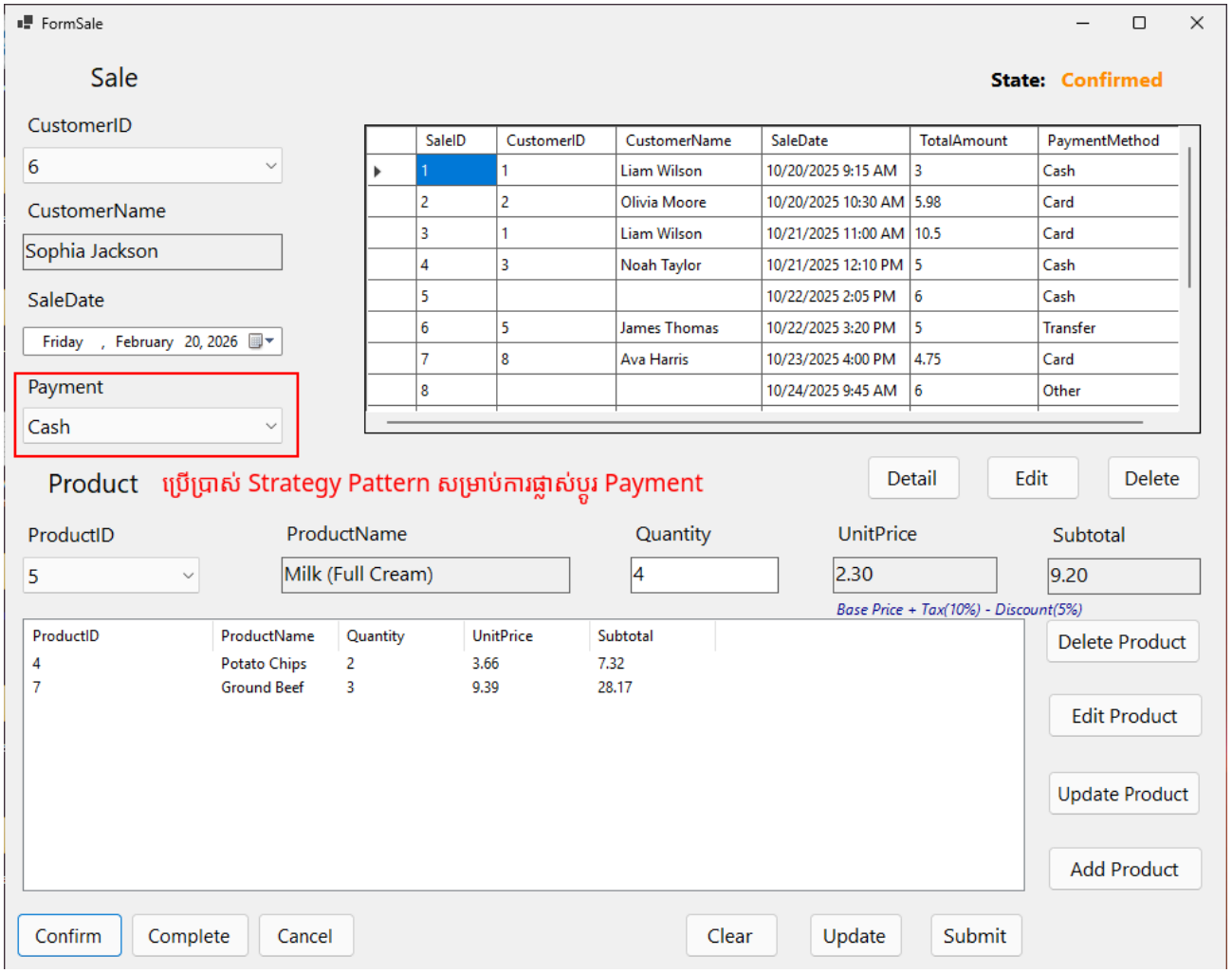
```
        {
            // Allowed in Draft state -- handled by the form
        }

        public void Confirm(OrderContext context)
        {
            context.TransitionTo(new ConfirmedState());
            MessageBox.Show("Order confirmed successfully.", "State
Change",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        public void Complete(OrderContext context)
        {
            MessageBox.Show("Cannot complete an order that hasn't been
confirmed yet.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }

        public void Cancel(OrderContext context)
        {
            context.TransitionTo(new CancelledState());
            MessageBox.Show("Order has been cancelled.", "State
Change",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }

    /// <summary>
    /// Confirmed State: Order is locked. No more item changes.
    /// Can transition to Completed or Cancelled.
    /// </summary>
    public class ConfirmedState : IOrderState
    {
        public string StateName => "Confirmed";

        public void AddItem(OrderContext context)
        {
            MessageBox.Show("Cannot add items to a confirmed order.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }
```

```
        public void Confirm(OrderContext context)
        {
            MessageBox.Show("Order is already confirmed.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }

        public void Complete(OrderContext context)
        {
            context.TransitionTo(new CompletedState());
            MessageBox.Show("Order completed and saved successfully.",
"State Change",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

        public void Cancel(OrderContext context)
        {
            context.TransitionTo(new CancelledState());
            MessageBox.Show("Confirmed order has been cancelled.",
"State Change",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }

    /// <summary>
    /// Completed State: Final state. No further changes allowed.
    /// </summary>
    public class CompletedState : IOrderState
    {
        public string StateName => "Completed";

        public void AddItem(OrderContext context)
        {
            MessageBox.Show("Cannot modify a completed order.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }

        public void Confirm(OrderContext context)
        {
            MessageBox.Show("Order is already completed.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }
```

```
        public void Complete(OrderContext context)
        {
            MessageBox.Show("Order is already completed.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }

        public void Cancel(OrderContext context)
        {
            MessageBox.Show("Cannot cancel a completed order.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }
    }

    /// <summary>
    /// Cancelled State: Terminal state. No further changes allowed.
    /// </summary>
    public class CancelledState : IOrderState
    {
        public string StateName => "Cancelled";

        public void AddItem(OrderContext context)
        {
            MessageBox.Show("Cannot modify a cancelled order.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }

        public void Confirm(OrderContext context)
        {
            MessageBox.Show("Cannot confirm a cancelled order.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }

        public void Complete(OrderContext context)
        {
            MessageBox.Show("Cannot complete a cancelled order.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }
```

```
        public void Cancel(OrderContext context)
        {
            MessageBox.Show("Order is already cancelled.",
                "Invalid Action", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        }
    }
}
```

19. Strategy Pattern



19.1 ការអនុវត្ត Strategy Pattern សម្រាប់ការទូទាត់ប្រាក់ (Payment Strategy)

Strategy Pattern ត្រូវបានប្រើដើម្បីកំណត់នូវ "បណ្តុំនៃវិធីសាស្ត្រ" (Algorithms) ហើយធ្វើឱ្យពួកវាអាចផ្លាស់ប្តូរគ្នាបាននៅពេល Runtime។ ក្នុងរូបភាពនេះ វាត្រូវបានអនុវត្តទៅលើផ្នែក Payment (ការទូទាត់)។

19.1.1 គោលបំណងសំខាន់ (Main Purpose)

- ការផ្លាស់ប្តូរវិធីសាស្ត្រទូទាត់: អនុញ្ញាតឱ្យប្រព័ន្ធជ្រើសរើសវិធីសាស្ត្រទូទាត់ខុសៗគ្នា (ដូចជា Cash, Card, Transfer) ដោយមិនចាំបាច់សរសេរកូដ `if-else` ច្រើនស្មុគស្មាញនៅក្នុង Logic ចម្បង។
- ភាពងាយស្រួលក្នុងការបន្ថែម: ប្រសិនបើថ្ងៃក្រោយអ្នកចង់បន្ថែមវិធីទូទាត់ថ្មី (ឧទាហរណ៍៖ ABA KHQR) អ្នកគ្រាន់តែបង្កើត Strategy ថ្មីមួយមកជំនួស
  ដោយមិនចាំបាច់កែប្រែកូដចាស់នៅក្នុង Form ឡើយ។

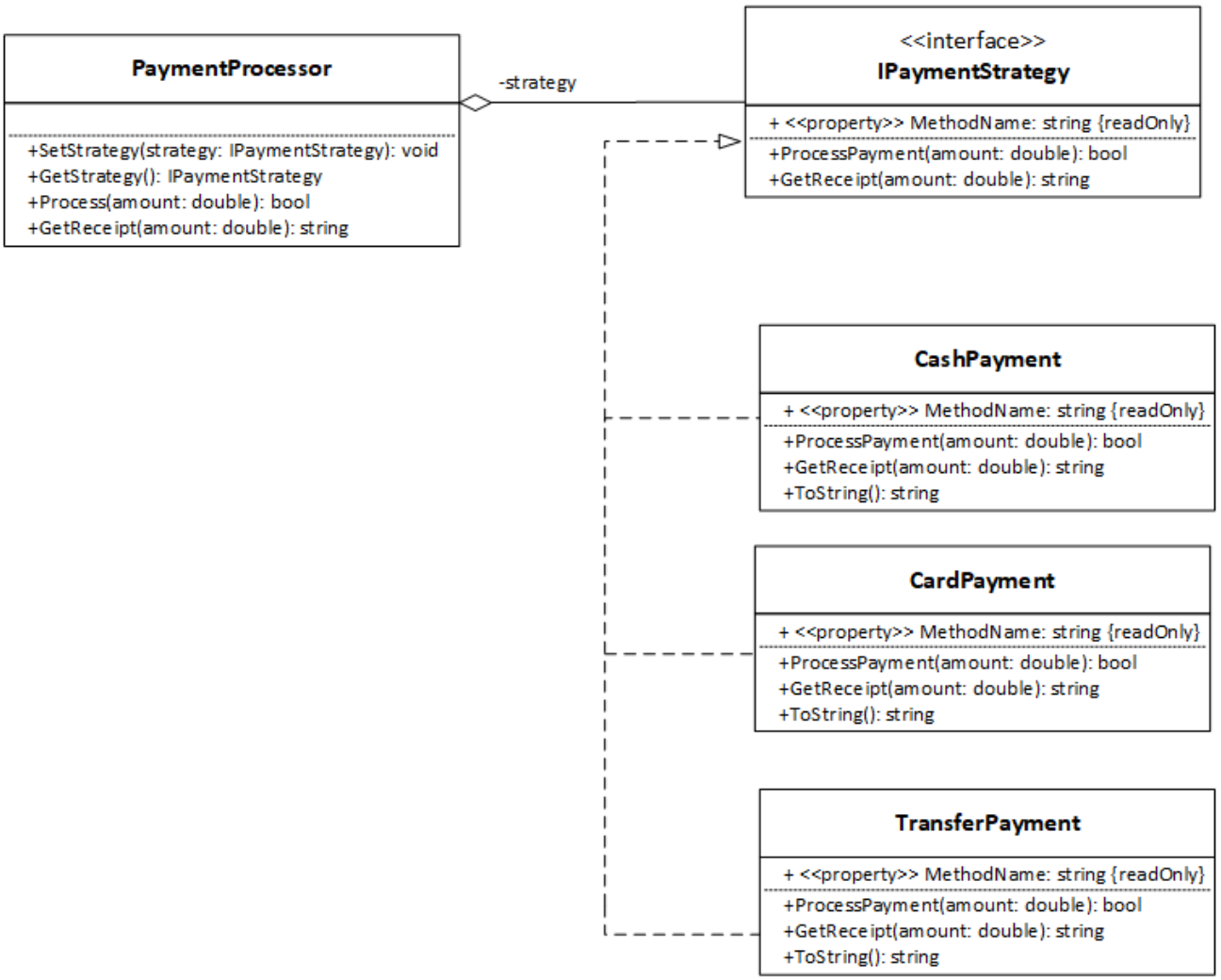
19.1.2 របៀបដែលវាដំណើរការក្នុង Form នេះ

- Context (FormSale): នៅពេលអ្នកប្រើប្រាស់ជ្រើសរើសប្រភេទក្នុងប្រអប់ Payment (Cash, Card, Transfer, etc.)។
- Strategy Selection: កម្មវិធីនឹងប្តូរ Object នៃវិធីសាស្ត្រទូទាត់ទៅតាមជម្រើសនោះ។ ឧទាហរណ៍៖ ប្រសិនបើជ្រើសរើស "Card" វានឹងប្រើប្រាស់ `CardPayment` ដើម្បីចាត់ចែងការទូទាត់។
- Execution: នៅពេលចុចប៊ូតុង Submit ប្រព័ន្ធនឹងហៅ Method `.ProcessPayment()` នៃ Strategy ដែលបានជ្រើសរើស ដើម្បីបញ្ចប់ប្រតិបត្តិការ។

គុណសម្បត្តិក្នុងការប្រើ Pattern នេះ៖

- បំបែក Logic ដាច់ដោយឡែក: កូដសម្រាប់ទូទាត់តាមកាត និងតាមសាច់ប្រាក់សុទ្ធ ស្ថិតនៅដាច់ពីគ្នា ធ្វើឱ្យងាយស្រួលគ្រប់គ្រង និង Validation។
- ភាពបត់បែន: អ្នកអាចដូរវិធីសាស្ត្រទូទាត់បានភ្លាមៗរាល់ពេលលក់ ដោយគ្រាន់តែរើសតាម Dropdown List។

19.2 UML Class Diagram



19.3 Implementation Code

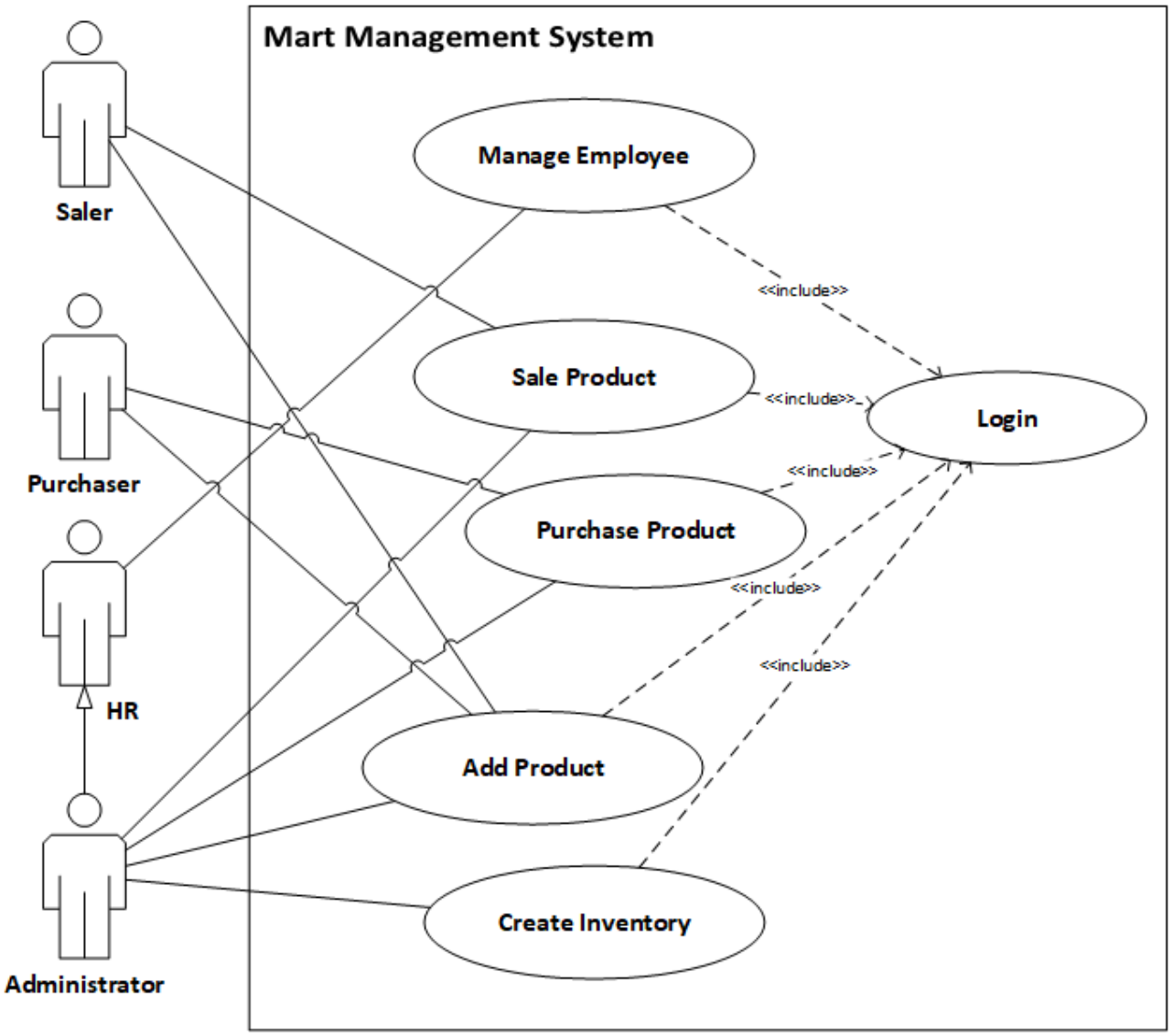
```
using System.Windows.Forms;

namespace mart_management
{
    /// <summary>
    /// Strategy Pattern: Interface for interchangeable payment
methods.
    /// Each strategy encapsulates its own payment processing logic.
    /// </summary>
    public interface IPaymentStrategy
    {
        string MethodName { get; }
        bool ProcessPayment(double amount);
        string GetReceipt(double amount);
    }
```

```
    /// <summary>
    /// Concrete Strategy: Cash payment processing.
    /// </summary>
    public class CashPayment : IPaymentStrategy
    {
        public string MethodName => "Cash";

        public bool ProcessPayment(double amount)
        {
            // Cash payment is always accepted
            return true;
        }

        public string GetReceipt(double amount)
        {
            return $"[CASH RECEIPT]\nAmount Paid: ${amount:F2}\nPayment
Method: Cash\nStatus: Completed";
        }

        public override string ToString() => MethodName;
    }

    /// <summary>
    /// Concrete Strategy: Card payment processing.
    /// </summary>
    public class CardPayment : IPaymentStrategy
    {
        public string MethodName => "Card";

        public bool ProcessPayment(double amount)
        {
            // Simulate card processing -- succeeds for amounts > 0
            if (amount <= 0)
            {
                MessageBox.Show("Card payment requires a positive
amount.",
                    "Card Error", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return false;
            }
            return true;
        }

        public string GetReceipt(double amount)
```

```
        {
            return $"[CARD RECEIPT]\nAmount Charged:
${amount:F2}\nPayment Method: Credit/Debit Card\nStatus: Approved";
        }

        public override string ToString() => MethodName;
    }

    /// <summary>
    /// Concrete Strategy: Bank transfer payment processing.
    /// </summary>
    public class TransferPayment : IPaymentStrategy
    {
        public string MethodName => "Transfer";

        public bool ProcessPayment(double amount)
        {
            // Simulate transfer processing -- succeeds for amounts > 0
            if (amount <= 0)
            {
                MessageBox.Show("Transfer requires a positive amount.",
                    "Transfer Error", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return false;
            }
            return true;
        }

        public string GetReceipt(double amount)
        {
            return $"[TRANSFER RECEIPT]\nAmount Transferred:
${amount:F2}\nPayment Method: Bank Transfer\nStatus: Processed";
        }

        public override string ToString() => MethodName;
    }

    /// <summary>
    /// Context class: Uses the selected payment strategy to process
payments.
    /// </summary>
    public class PaymentProcessor
    {
        private IPaymentStrategy? _strategy;
```

```
        public void SetStrategy(IPaymentStrategy strategy)
        {
            _strategy = strategy;
        }

        public IPaymentStrategy? GetStrategy() => _strategy;

        public bool Process(double amount)
        {
            if (_strategy == null)
            {
                MessageBox.Show("Please select a payment method
first.",
                    "Payment Error", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return false;
            }
            return _strategy.ProcessPayment(amount);
        }

        public string GetReceipt(double amount)
        {
            if (_strategy == null) return "No payment method
selected.";
            return _strategy.GetReceipt(amount);
        }
    }
}
```

20. Use Case Diagram



រូបនេះបង្ហាញពីរបៀបដែលអ្នកប្រើ (Actors) ផ្សេងៗអាចធ្វើសកម្មភាព (Use Cases) នៅក្នុងប្រព័ន្ធគ្រប់គ្រងផ្សារ។ ប្រព័ន្ធនេះមានមុខងារចម្បងៗដូចជា ការលក់ផលិតផល ការទិញផលិតផល ការបន្ថែមផលិតផល ការគ្រប់គ្រងបុគ្គលិក និងការបង្កើតសារពើភ័ណ្ឌ។

20.1 អ្នកប្រើ (Actors)

- Saler (អ្នកលក់) អាចធ្វើសកម្មភាព៖
    - Sale Product (លក់ផលិតផល)
    - Add Product (បន្ថែមផលិតផល)
    - Login (ត្រូវចូលប្រព័ន្ធជាមុន)

- Purchaser (អ្នកទិញ) អាចធ្វើសកម្មភាព៖
    - Purchase Product (ទិញផលិតផល)
    - Add Product (បន្ថែមផលិតផល)
    - Login (ត្រូវចូលប្រព័ន្ធជាមុន)
- HR (ធនធានមនុស្ស) អាចធ្វើសកម្មភាព៖
    - Manage Employee (គ្រប់គ្រងបុគ្គលិក)
    - Login (ត្រូវចូលប្រព័ន្ធជាមុន)
- Administrator (អ្នកគ្រប់គ្រងប្រព័ន្ធ) អាចធ្វើសកម្មភាពទាំងអស់៖
    - Manage Employee (គ្រប់គ្រងបុគ្គលិក)
    - Sale Product (លក់ផលិតផល)
    - Purchase Product (ទិញផលិតផល)
    - Add Product (បន្ថែមផលិតផល)
    - Create Inventory (បង្កើតសារពើភ័ណ្ឌ)
    - Login (ចូលប្រព័ន្ធ)

20.2 សកម្មភាព (Use Cases)

- Login (ចូលប្រព័ន្ធ) ជា Use Case ដែលត្រូវ «include» នៅក្នុងគ្រប់សកម្មភាពផ្សេងទៀត។ នោះមានន័យថា មុននឹងអាចធ្វើសកម្មភាពណាមួយ អ្នកប្រើត្រូវចូលប្រព័ន្ធជាមុនសិន។
- Sale Product (លក់ផលិតផល) អនុញ្ញាតឲ្យអ្នកលក់ ឬអ្នកគ្រប់គ្រងលក់ផលិតផល។ សកម្មភាពនេះ include «Login»។
- Purchase Product (ទិញផលិតផល) អនុញ្ញាតឲ្យអ្នកទិញ ឬអ្នកគ្រប់គ្រងទិញផលិតផលពីអ្នកផ្គត់ផ្គង់។ សកម្មភាពនេះ include «Login»។
- Add Product (បន្ថែមផលិតផល) អនុញ្ញាតឲ្យបន្ថែមផលិតផលថ្មីទៅក្នុងប្រព័ន្ធ។ សកម្មភាពនេះ include «Login»។
- Manage Employee (គ្រប់គ្រងបុគ្គលិក) អនុញ្ញាតឲ្យ HR ឬ Administrator គ្រប់គ្រងព័ត៌មានបុគ្គលិក។ សកម្មភាពនេះ include «Login»។
- Create Inventory (បង្កើតសារពើភ័ណ្ឌ) អនុញ្ញាតឲ្យ Administrator បង្កើត ឬកែប្រែសារពើភ័ណ្ឌ។ សកម្មភាពនេះ include «Login»។

20.3 ទំនាក់ទំនង «include»

ទំនាក់ទំនង «include» បង្ហាញថាសកម្មភាពទាំងអស់ត្រូវអនុវត្ត Login ជាមុន មុនពេលអាចធ្វើសកម្មភាពនោះ។ នោះបង្ហាញថា “Login” គឺជាផ្នែក ចាំបាច់ នៃ Use Case ផ្សេងទៀតទាំងអស់។

21. Entity Relationship Diagram

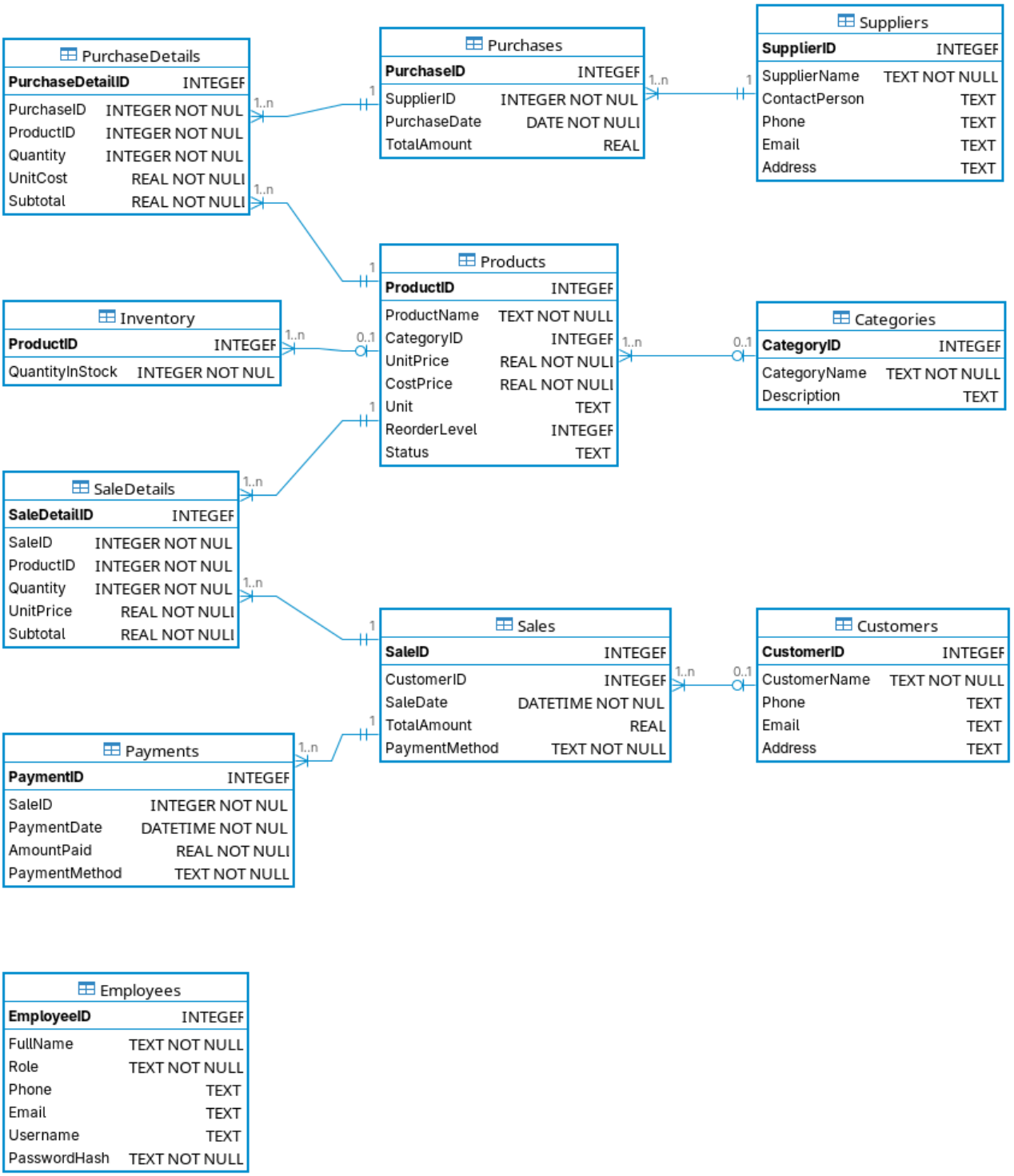


Diagram នេះបង្ហាញពី តារាង (Tables) និង ទំនាក់ទំនង (Relationships) នៅក្នុងមូលដ្ឋានទិន្នន័យរបស់ ប្រព័ន្ធគ្រប់គ្រងផ្សារ (Mart Management System)។ លើសពីនោះទៀត Diagram នេះក៏បង្ហាញយ៉ាងច្បាស់ពីរបៀបដែល ការទិញ ការលក់ សារពើភ័ណ្ឌ អតិថិជន អ្នកផ្គត់ផ្គង់ និង បុគ្គលិក មានទំនាក់ទំនងគ្នា ដើម្បីបង្កើតប្រព័ន្ធគ្រប់គ្រងដែលមានប្រសិទ្ធភាព

អាចតាមដានលំហូរផលិតផលពី ដំណាក់កាលនៃការទិញ ឆ្ពោះទៅការលក់
និងការទូទាត់បានយ៉ាងពេញលេញ និងមានភាពជាក់លាក់។

21.1 Suppliers (អ្នកផ្គត់ផ្គង់)

ផ្ទុកព័ត៌មានអំពីអ្នកផ្គត់ផ្គង់ទំនិញ។
Fields:

- SupplierID *(Primary Key)*
- SupplierName
- ContactPerson
- Phone
- Email
- Address

Relationship:
Supplier ត្រូវបានភ្ជាប់ទៅ Purchases (ការទិញ) តាមរយៈ SupplierID។ មួយ Supplier អាចមាន Purchase ច្រើន។

21.2 Purchases (ការទិញផលិតផល)

ផ្ទុកព័ត៌មានអំពីការទិញផលិតផលពីអ្នកផ្គត់ផ្គង់។
Fields:

- PurchaseID *(Primary Key)*
- SupplierID *(Foreign Key → Suppliers)*
- PurchaseDate
- TotalAmount

Relationship:

- មួយ Purchase មាន PurchaseDetails ច្រើន (1 → n)
- ត្រូវភ្ជាប់ទៅ Supplier តែមួយ (n → 1)

21.3 PurchaseDetails (ព័ត៌មានលម្អិតការទិញ)

ផ្ទុកព័ត៌មានអំពីផលិតផលដែលបានទិញក្នុងការទិញនីមួយៗ។
Fields:

- PurchaseDetailID *(Primary Key)*
- PurchaseID *(FK → Purchases)*
- ProductID *(FK → Products)*
- Quantity
- UnitCost
- Subtotal

Relationship:

- មួយ Purchase មាន PurchaseDetails ច្រើន
- មួយ Product អាចមានក្នុង PurchaseDetails ច្រើន

21.4 Products (ផលិតផល)

ផ្ទុកព័ត៌មានអំពីផលិតផលក្នុងផ្សារ។
Fields:

- ProductID *(Primary Key)*
- ProductName
- CategoryID *(FK → Categories)*
- UnitPrice
- CostPrice
- Unit
- ReorderLevel
- Status

Relationship:

- មួយ Category មាន Products ច្រើន
- Products ត្រូវបានប្រើក្នុង PurchaseDetails, SaleDetails, Inventory

21.5 Categories (ប្រភេទផលិតផល)

ចាត់ថ្នាក់ផលិតផលជាក្រុម។
Fields:

- CategoryID *(Primary Key)*

- CategoryName
- Description

Relationship:
Category ត្រូវបានភ្ជាប់ទៅ Products (1 → n)

21.6 Inventory (សារពើភ័ណ្ឌ)

បង្ហាញពីចំនួនផលិតផលនៅក្នុងស្តុក។
Fields:

- ProductID *(FK → Products)*
- QuantityInStock

Relationship:
មួយ Product មានមួយ Inventory (1 → 1)

21.7 Sales (ការលក់ផលិតផល)

ផ្ទុកព័ត៌មានអំពីការលក់ផលិតផលទៅអតិថិជន។
Fields:

- SaleID *(Primary Key)*
- CustomerID *(FK → Customers)*
- SaleDate
- TotalAmount
- PaymentMethod

Relationship:

- មួយ Sale មាន SaleDetails ច្រើន (1 → n)
- មួយ Customer អាចមាន Sales ច្រើន

21.8 SaleDetails (ព័ត៌មានលម្អិតការលក់)

ផ្ទុកព័ត៌មានអំពីផលិតផលដែលបានលក់ក្នុងការលក់មួយ។
Fields:

- SaleDetailID *(Primary Key)*

- SaleID *(FK → Sales)*
- ProductID *(FK → Products)*
- Quantity
- UnitPrice
- Subtotal

Relationship:

- មួយ Sale មាន SaleDetails ច្រើន
- មួយ Product អាចលក់បានច្រើនដង

21.9 Payments (ការទូទាត់)

ផ្ទុកព័ត៌មានអំពីការទូទាត់របស់អតិថិជនសម្រាប់ការលក់។
Fields:

- PaymentID *(Primary Key)*
- SaleID *(FK → Sales)*
- PaymentDate
- AmountPaid
- PaymentMethod

Relationship:

- មួយ Sale មាន Payments ច្រើន (1 → n)

21.10 Customers (អតិថិជន)

ផ្ទុកព័ត៌មានអតិថិជន។
Fields:

- CustomerID *(Primary Key)*
- CustomerName
- Phone
- Email
- Address

Relationship:
មួយ Customer អាចមាន Sales ច្រើន (1 → n)

21.11 Employees (បុគ្គលិក)

ផ្ទុកព័ត៌មានអ្នកធ្វើការ ឬអ្នកប្រើប្រព័ន្ធ។
Fields:

- EmployeeID *(Primary Key)*
- FullName
- Role
- Phone
- Email
- Username
- PasswordHash

Purpose:
ប្រើសម្រាប់ចូលប្រព័ន្ធ និងកំណត់តួនាទី (Role) របស់អ្នកប្រើ (ឧ. Administrator, HR, Saler, Purchaser)។

22. Conclusion (សន្និដ្ឋាន)

តាមរយៈការអភិវឌ្ឍប្រព័ន្ធ Mart Management System នេះ យើងអាចសន្និដ្ឋានបានថា ប្រព័ន្ធនេះអាចជួយដល់សហគ្រាស ឬអាជីវកម្មផ្សារ ក្នុងការគ្រប់គ្រងទិន្នន័យជាប្រព័ន្ធ និងកាត់បន្ថយកំហុសមនុស្ស។

ប្រព័ន្ធនេះបានបង្ហាញពីការអនុវត្តន៍យ៉ាងជាក់ស្តែងនៃគោលការណ៍ **Object-Oriented Programming (OOP)** ដោយប្រើ **UML Class Diagram** ដើម្បីបង្ហាញទំនាក់ទំនងរវាង Classes ផ្សេងៗ និងការប្រើប្រាស់ **Software Design Pattern** ដូចជា *Builder, Singleton, Composite, Decorator, State,* និង *Strategy*។ ការរចនាបែបនេះ ធ្វើឱ្យប្រព័ន្ធមានភាពងាយស្រួលក្នុងការគ្រប់គ្រងវត្ថុ (Objects) ដែលមានភាពស្មុគស្មាញ និងផ្តល់នូវភាពបត់បែនខ្ពស់ក្នុងការរក្សាទុក កែប្រែ ឬពង្រីកមុខងារថ្មីៗ។

សៀវភៅនេះមិនត្រឹមតែជាគំរូសម្រាប់សិក្សាផ្នែកទ្រឹស្តីទេ ប៉ុន្តែជាគំរូអនុវត្តន៍ពិតប្រាកដសម្រាប់ការរចនាប្រព័ន្ធព័ត៌មានដែលអាចប្រើបានក្នុងពេលអនាគត។ តាមរយៈការអនុវត្តប្រព័ន្ធនេះ អ្នកអាចទទួលបានបទពិសោធន៍ក្នុងការរចនា កូដ និងគ្រប់គ្រងប្រព័ន្ធឌីជីថលដោយប្រើគោលការណ៍វិទ្យាសាស្ត្រសមស្រប។