Стеценко Ілля Валерійович

Студент групи ШІ

Формальна специфікація системи «Снек-автомат»

Був використаний **моделе-орієнтовний метод** специфікації, тобто описано аспекти специфікуємої системи за допомогою аксіом, що властиві специфікаціям, орієнтованим на властивостям.

У даній роботі використано **модельно-орієнтований метод специфікації**, тобто аспекти системи описано за допомогою заданої моделі стану та властивостей (аксіом), характерних для специфікацій, орієнтованих на властивості. Тип специфікації обрано **поведінковий** – описано обмеження на поведінку об’єкта специфікації (в нашому випадку – снек-автомат), зосереджено на його функціональних можливостях, правилах безпеки та виконання операцій. Частково специфікацію можна віднести й до **структурної**, оскільки описано внутрішню будову об’єкта: асортимент снеків (склад і кількість), модуль для оплати карткою та механізм видачі товару.

Перейдемо до опису системи **«Снек-автомат»**. Серед найбільш практичних функцій, що реалізує дана система, розглянемо такі основні операції:

- **Функція «Обрати снек»** – користувач вводить номер позиції товару для вибору бажаного снека.

- **Функція «Оплатити снек»** – оплата вибраного снека **карткою** (інших способів оплати не передбачено); виконується лише після успішного вибору товару і за умови його наявності.

- **Функція «Видати снек»** – фізична видача товару; можлива тільки після підтвердження факту оплати.

Визначимо умови, якими керуються ці функції (правила безпеки й коректності роботи автомату):

- Не можна **обрати, оплатити чи видати** снек, якщо його **немає в наявності** (в автоматі відсутня хоча б 1 одиниця даного товару).

- Оплата можлива **лише карткою** – автомат не приймає готівку чи інші способи оплати.

- **Видача снека** здійснюється **тільки після успішної оплати** (неоплачений товар не може бути видано зі складу).

Зібравши всі необхідні умови, можемо приступити до формального опису функції за допомогою псевдо-коду:

Як може виглядати функція «Обрати снек»

Функція: Обрати снек

Вхідні дані: вибраний номер позиції (ідентифікатор) товару, снек не обраний або раніше обраний снек не оплачений (тобто можемо переобрати інший ідентифікатор)

Якщо обраного снека немає в наявності:

-> Обрати неможливо

-> Зберігаємо вибраний снек (фіксуємо вибір користувача)

Функція: Оплатити снек

Вхідні дані: спосіб оплати (картка), снек обраний та снек не оплачено

Якщо жоден снек не вибрано або вибраного снека немає в наявності:

-> Оплатити неможливо

Якщо спосіб оплати != картка:

-> Оплатити неможливо

-> Списуємо з картки суму вартості обраного снека

-> Збільшуємо сумарну виручку на вартість снека

-> Позначаємо оплату як успішну (підтверджуємо оплату)

Функція: Видати снек

Вхідні дані: снек обраний та оплачений.

Якщо оплату не здійснено (не було успішної оплати):

-> Видати неможливо

Якщо обраного снека немає в наявності:

-> Видати неможливо

-> Відчиняємо лоток видачі

-> Видаємо вибраний снек користувачу

-> Зменшуємо кількість вибраного снека на 1 в автоматі

Після того, як ми описали методи, можна перейти до їхньої реалізації.

Короткий опис проекту:

Мова програмування – *Dafny*.

Середовище розробки – *VS Code.*

Реалізація проекту:

У основу всього проекту ліг створений загальний клас, що має назву SnackMachine.

У ньому зберігається інформація про наявні снеки (ціни та кількість), поточний обраний снек чи здійснена оплата та поточна виручка.

```
var inventory: map<int,int>
   var prices: map<int,int>
   var totalRevenue: int
   var selectedSnack: int
   var paymentDone: bool
```

Конструктор, в який ми передаємо початкові значення для цих полів:

```
constructor(initInventory: map<int,int>, initPrices: map<int,int>,
initRevenue: int)
       requires forall k :: k in initInventory ==> k > 0
       requires forall k :: k in initPrices ==> k > 0
       // Ensure all inventory items have prices
       requires forall k :: k in initInventory ==> k in initPrices

       ensures inventory == initInventory
       ensures prices == initPrices
       ensures totalRevenue == initRevenue
       ensures selectedSnack == 0
       ensures !paymentDone
   {
       inventory := initInventory;
       prices := initPrices;
       totalRevenue := initRevenue;
       selectedSnack := 0;         // Nothing selected initially
       paymentDone := false;       // No payment initially
   }
```

Метод «Select Snack»

```
method SelectSnack(snackID: int) returns (success: bool)
       requires forall k :: k in this.inventory ==> k > 0 // each key in
greater than 0
       requires forall k :: k in this.prices ==> k > 0 // each key in
greater than 0
       requires !this.paymentDone // in initial state
       requires snackID > 0 // selected snack in greater than 0
       modifies this
       ensures success ==> this.inventory == old(this.inventory) &&
this.prices == old(this.prices)
       ensures success ==> this.selectedSnack == snackID
       ensures success ==> snackID in this.inventory && snackID in
this.prices
       ensures !success ==> this.selectedSnack == old(this.selectedSnack)
       ensures !this.paymentDone
   {
       // Check if the selected snack exists and is available
       if !(snackID in inventory) || inventory[snackID] == 0 {
           // Selection not possible
           success := false;
           return;
       }
```

```
        if !(snackID in this.inventory)
        || !(snackID in this.prices) {
            // Payment not possible
            return false;
        }
        // Store snack selection
        this.selectedSnack := snackID;
        // Reset payment flag (start of new transaction)
        this.paymentDone := false;
        return true;
    }
```

Метод «Pay for snack»

```
// Method "Pay for Snack"
    method PayForSnack(card: bool) returns (success: bool)
        requires forall k :: k in this.inventory ==> k > 0 // each key in
greater than 0
        requires forall k :: k in this.prices ==> k > 0 // each key in
greater than 0
        requires this.selectedSnack > 0 && !this.paymentDone
        requires this.selectedSnack in this.inventory && this.selectedSnack
in this.prices
        modifies this

        ensures this.selectedSnack == old(this.selectedSnack)
        ensures !success ==> !this.paymentDone
        ensures success ==> this.paymentDone
        ensures success ==> this.inventory == old(this.inventory) &&
this.prices == old(this.prices)
    {
        if (this.selectedSnack <= 0 || this.paymentDone) {
            success := false;
            return success;
        }
        // Check: is a snack selected and available
        if !(this.selectedSnack in this.inventory) ||
this.inventory[this.selectedSnack] == 0 {
            // Payment not possible
            success := false;
            // this.selectedSnack
            return success;
        }
        // Payment only possible by card
```

```
        if !card {
            // Payment not possible (non-card method not accepted)
            success := false;
            return success;
        }
        // Get the price of the selected snack
        var found, price := this.TryGetPrice();
        if !found {
            success := false;
            return success;
        }
        // Increase total revenue by the payment amount
        this.totalRevenue := this.totalRevenue + price;
        // Record successful payment
        this.paymentDone := true;
        success := true;
        return success;
    }
```

Метод «Dispense snack»

```
method DispenseSnack() returns (success: bool)
        requires this.selectedSnack > 0 && this.paymentDone
        requires this.selectedSnack in this.inventory
        requires this.inventory[this.selectedSnack] > 0
        modifies this
        ensures !this.paymentDone
        ensures this.selectedSnack == -1

        ensures success ==> !this.paymentDone && this.selectedSnack == -1
        ensures !success ==> this.paymentDone == old(this.paymentDone) &&
this.selectedSnack == old(this.selectedSnack)
    {
        if (!this.paymentDone) {
            return false;
        }
        var current := inventory[selectedSnack];
        var newCount := current - 1;
        this.inventory := inventory[selectedSnack := newCount];
        // Reset selection and payment state after dispensing
        this.selectedSnack := -1;
        this.paymentDone := false;

        return true;
```

```
    }
```

Для симуляції поведінки цієї системи напишемо окремо функцію Main, що буде вхідною точкою для нашого проекту.

```
static method Main() {
        // Initialize machine: code 1 -> 5 units, 2 -> 3 units, 3 -> 0
units; prices: 1 -> 15, 2 -> 10, 3 -> 7
        var initInv := map[1 := 5, 2 := 3, 3 := 0];
        var initPrices := map[1 := 15, 2 := 10, 3 := 7];
        var machine := new SnackMachine(initInv, initPrices, 0);
        print "Initial quantity of snacks (code 2): ",
machine.inventory[2], "\n";
        print "Initial revenue: ", machine.totalRevenue, "\n";

        // User selects snack with code 2
        var selectionResult := machine.SelectSnack(2);
        print "Selection success (code 2): ", selectionResult, "\n";

        if (!selectionResult) {
            return;
        }

        // Payment for selected snack (by card)
        var paymentResult := machine.PayForSnack(true);
        print "Payment success: ", paymentResult, "\n";
        print "Revenue after payment: ", machine.totalRevenue, "\n";

        if (!paymentResult) {
            return;
        }

        // Dispensing paid snack
        var result := machine.DispenseSnack();
        if (result) {
            print "Dispensed successfully";
        } else {
            print "error occured";
        }
    }
```

У даному методі:

- Створюємо снек автомат з 3-ма снеками з відповідними кодами та цінами
- Потім обираємо снек 2.
- Оплачуємо його
- Видаємо снек

Переконаємось у цьому:

Initial quantity of snacks (code 2): 3

Initial revenue: 0

Selection success (code 2): true

Payment success: true

Revenue after payment: 10

Висновок: Отже, у цій частині першої контрольної роботи ми навчились диференціювати різні типи та методи формальної специфікації, застосовувати ці знання на конкретних системах для опису основного функціоналу, працювати з мовою Dafny.

Лістинг коду

```
class SnackMachine {
    var inventory: map<int,int>
    var prices: map<int,int>
    var totalRevenue: int
    var selectedSnack: int
    var paymentDone: bool

    constructor(initInventory: map<int,int>, initPrices: map<int,int>,
initRevenue: int)
        requires forall k :: k in initInventory ==> k > 0
        requires forall k :: k in initPrices ==> k > 0
        // Ensure all inventory items have prices
        requires forall k :: k in initInventory ==> k in initPrices

        ensures inventory == initInventory
        ensures prices == initPrices
        ensures totalRevenue == initRevenue
        ensures selectedSnack == 0
        ensures !paymentDone
    {
        inventory := initInventory;
        prices := initPrices;
        totalRevenue := initRevenue;
        selectedSnack := 0;         // Nothing selected initially
        paymentDone := false;       // No payment initially
    }

    // Method "Select Snack"
    method SelectSnack(snackID: int) returns (success: bool)
```

```
        requires forall k :: k in this.inventory ==> k > 0 // each key
in greater than 0

        requires forall k :: k in this.prices ==> k > 0 // each key in
greater than 0

        requires !this.paymentDone // in initial state

        requires snackID > 0 // selected snack in greater than 0

        modifies this

        ensures success ==> this.inventory == old(this.inventory) &&
this.prices == old(this.prices)

        ensures success ==> this.selectedSnack == snackID

        ensures success ==> snackID in this.inventory && snackID in
this.prices

        ensures !success ==> this.selectedSnack ==
old(this.selectedSnack)

        ensures !this.paymentDone

    {

        // Check if the selected snack exists and is available

        if !(snackID in inventory) || inventory[snackID] == 0 {

            // Selection not possible

            success := false;

            return;

        }

        if !(snackID in this.inventory)

        || !(snackID in this.prices) {

            // Payment not possible

            return false;

        }

        // Store snack selection

        this.selectedSnack := snackID;

        // Reset payment flag (start of new transaction)

        this.paymentDone := false;

        return true;
```

```
    }

    method TryGetPrice() returns (found: bool, price: int)
        ensures found <==> this.selectedSnack in this.prices
        ensures found ==> price == this.prices[this.selectedSnack]
        ensures !found ==> price == 0
    {
        if selectedSnack in prices {
            price := prices[selectedSnack];
            found := true;
        } else {
            price := 0; // Default value
            found := false;
        }
    }

    // Method "Pay for Snack"
    method PayForSnack(card: bool) returns (success: bool)
        requires forall k :: k in this.inventory ==> k > 0 // each key
in greater than 0
        requires forall k :: k in this.prices ==> k > 0 // each key in
greater than 0
        requires this.selectedSnack > 0 && !this.paymentDone
        requires this.selectedSnack in this.inventory &&
this.selectedSnack in this.prices
        modifies this

        ensures this.selectedSnack == old(this.selectedSnack)
        ensures !success ==> !this.paymentDone
        ensures success ==> this.paymentDone
        ensures success ==> this.inventory == old(this.inventory) &&
this.prices == old(this.prices)
```

```
    {
        if (this.selectedSnack <= 0 || this.paymentDone) {
            success := false;
            return success;
        }
        // Check: is a snack selected and available
        if !(this.selectedSnack in this.inventory) ||
this.inventory[this.selectedSnack] == 0 {
            // Payment not possible
            success := false;
            // this.selectedSnack
            return success;
        }
        // Payment only possible by card
        if !card {
            // Payment not possible (non-card method not accepted)
            success := false;
            return success;
        }
        // Get the price of the selected snack
        var found, price := this.TryGetPrice();
        if !found {
            success := false;
            return success;
        }
        // Increase total revenue by the payment amount
        this.totalRevenue := this.totalRevenue + price;
        // Record successful payment
        this.paymentDone := true;
        success := true;
        return success;
```

```
    }

    // Method "Dispense Snack"
    method DispenseSnack() returns (success: bool)
        requires this.selectedSnack > 0 && this.paymentDone
        requires this.selectedSnack in this.inventory
        requires this.inventory[this.selectedSnack] > 0
        modifies this
        ensures !this.paymentDone
        ensures this.selectedSnack == -1

        ensures success ==> !this.paymentDone && this.selectedSnack ==
-1
        ensures !success ==> this.paymentDone == old(this.paymentDone)
&& this.selectedSnack == old(this.selectedSnack)
    {
        if (!this.paymentDone) {
            return false;
        }
        var current := inventory[selectedSnack];
        var newCount := current - 1;
        this.inventory := inventory[selectedSnack := newCount];
        // Reset selection and payment state after dispensing
        this.selectedSnack := -1;
        this.paymentDone := false;

        return true;
    }
}

class Tester {
```

```
   static method Main() {
       // Initialize machine: code 1 -> 5 units, 2 -> 3 units, 3 -> 0
units; prices: 1 -> 15, 2 -> 10, 3 -> 7
       var initInv := map[1 := 5, 2 := 3, 3 := 0];
       var initPrices := map[1 := 15, 2 := 10, 3 := 7];
       var machine := new SnackMachine(initInv, initPrices, 0);
       print "Initial quantity of snacks (code 2): ",
machine.inventory[2], "\n";
       print "Initial revenue: ", machine.totalRevenue, "\n";

       // User selects snack with code 2
       var selectionResult := machine.SelectSnack(2);
       print "Selection success (code 2): ", selectionResult, "\n";

       if (!selectionResult) {
           return;
       }

       // Payment for selected snack (by card)
       var paymentResult := machine.PayForSnack(true);
       print "Payment success: ", paymentResult, "\n";
       print "Revenue after payment: ", machine.totalRevenue, "\n";

       if (!paymentResult) {
           return;
       }

       // Dispensing paid snack
       var result := machine.DispenseSnack();
       if (result) {
           print "Dispensed successfully";
```

```
        } else {
            print "error occured";
        }
    }
}
```