i219 Software Design Methodology
8. Dynamic modeling 1

Kazuhiro Ogata (JAIST)

Outline of lecture

- Use case
- Use case diagram
- State (machine) diagram
- Sequence diagram
- Class diagram of vending machine
- Vending machine in Java

Use case (1)

Let us design a vending machine that sells milk, juice & coffee whose costs are 40Yen, 80Yen & 120Yen, and that accepts 10Yen, 50Yen & 100Yen coins.

First think about some of its use cases.

Each use case is a collection of scenarios that describe in which way users interact with a system to be developed to achieve some purpose.

Use case (1)

use case name

one main success & two exceptional scenarios

Insert Coin

Main Success Scenario:
1. User inserts 10Yen, 50Yen or 100Yen coin
2. System records it

Extensions:
2a: The inserted one is different from those three coins
.1: System lets user know what coins can be accepted

one main success & one exceptional scenarios

Select Item

Main Success Scenario:
1. User selects milk, juice or coffee
2. System delivers the item and subtracts its cost from the coins inserted

Extensions:
2a: The selected one is different from those three items
.1: System lets user know what items can be selected
2b: The coins inserted are not enough for the item
.1: System lets user know it

Use case (2)

Show Items

Main Success Scenario:
1. User asks system to show items bought
2. System displays them

Show Usage

Main Success Scenario:
1. User asks system to let him/her know how to use system
2. System displays the usage

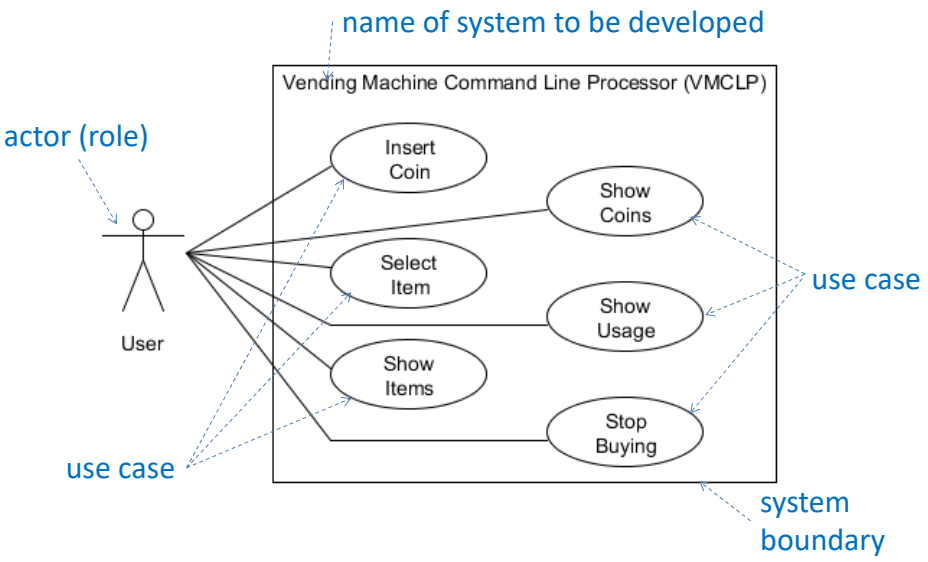
Show Coins

Main Success Scenario:
1. User asks system to show coins inserted
2. System displays them

Stop Buying

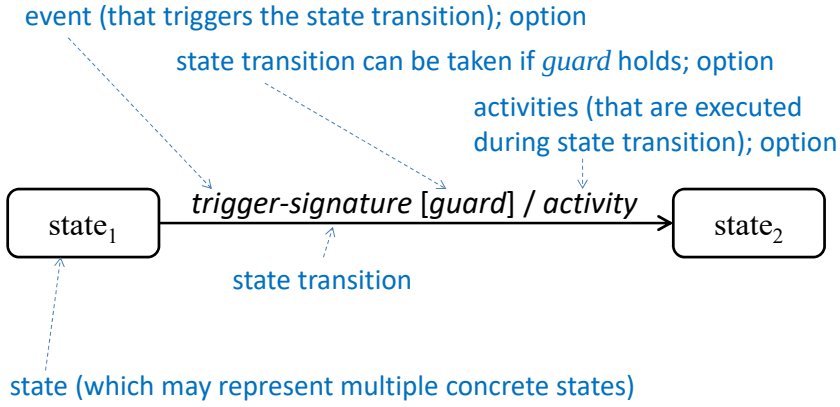
Main Success Scenario:
1. User lets system know he/she will stop buying
2. System displays items bought & returns change

Use case diagram

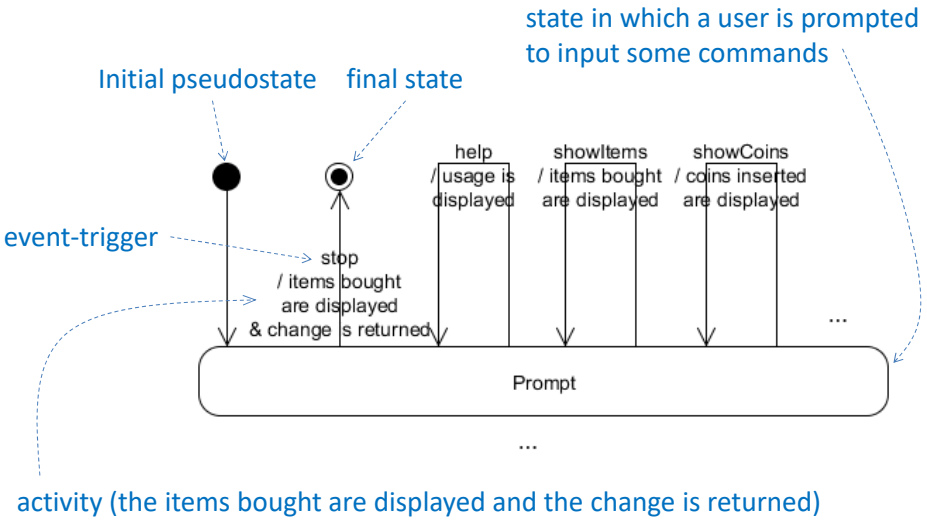


State diagram (1)

A state (machine) diagram describes the behavior of (a single object of) a class



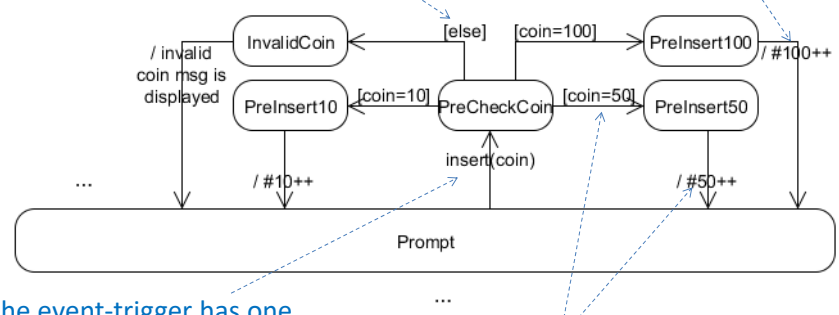
State diagram (2)



State diagram (3)

the number of 100Yen coins in Cash Box is incremented

coin ≠ 10 & coin ≠ 50 & coin ≠ 100



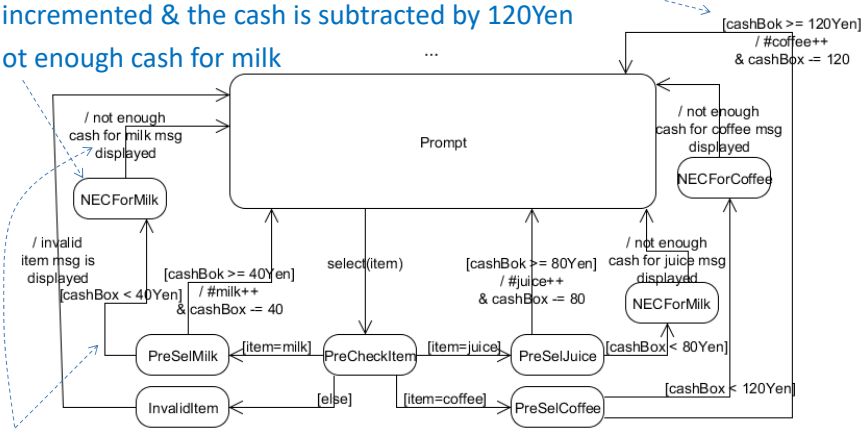
the event-trigger has one parameter coin

the two state transitions can be combined:
[coin=50] / #50++

State diagram (4)

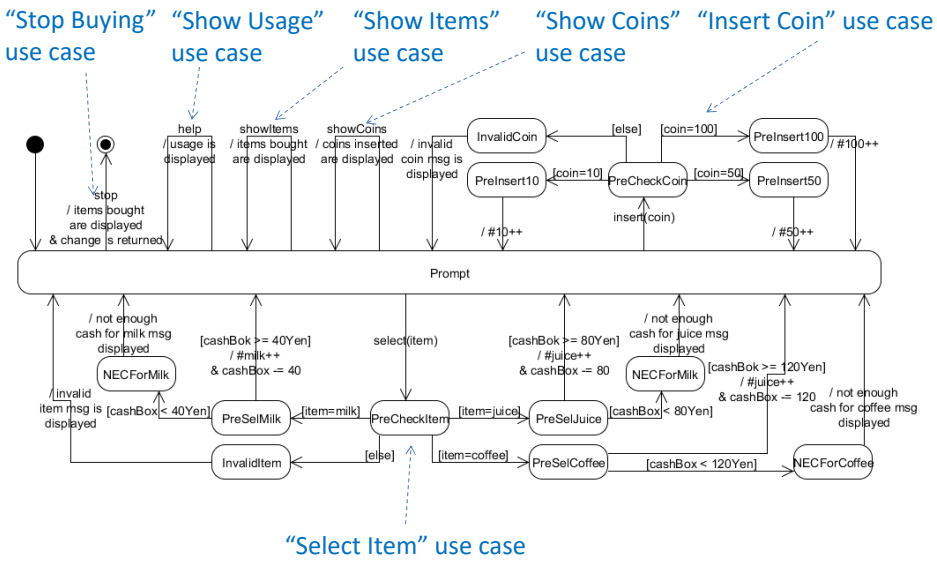
if the cash in the cash box is greater than or equal to 120Yen, then the number of coffee bought is incremented & the cash is subtracted by 120Yen

not enough cash for milk



can be combined:
[cashBox < 40Yen] / not enough cash for milk msg is displayed

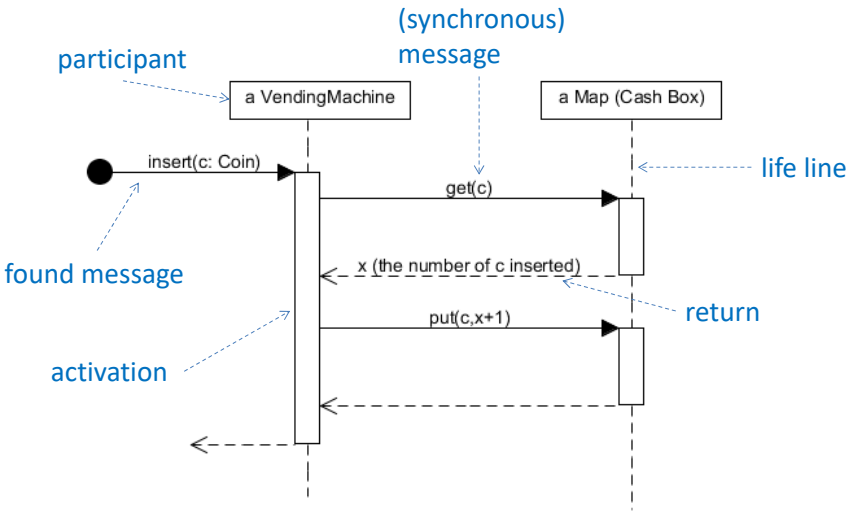
State diagram (5)



Sequence diagram (1)

A sequence diagram describes interaction among participants such as objects such that a participant sends a message to another participant and waits for some result.

Sequence diagram (2)

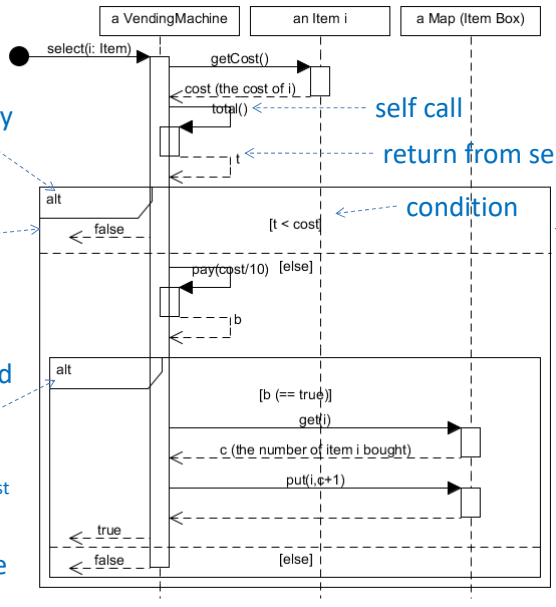


Sequence diagram (3)

alternative multiple fragments; the 1st fragment whose condition holds is only executed

if t < cost holds, then the 1st fragment is executed; otherwise, the 2nd one is executed

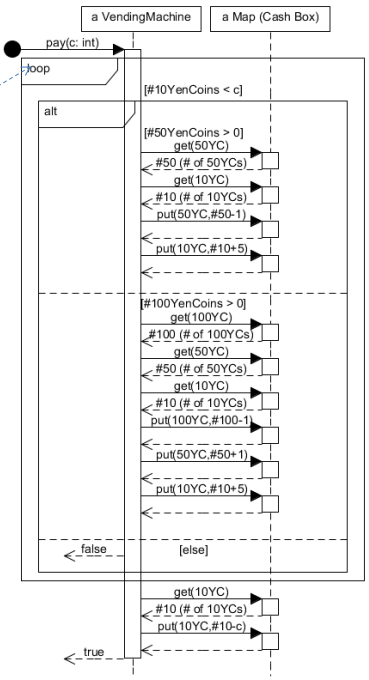
if b is true, then the 1st fragment is executed; otherwise, the 2nd one is executed



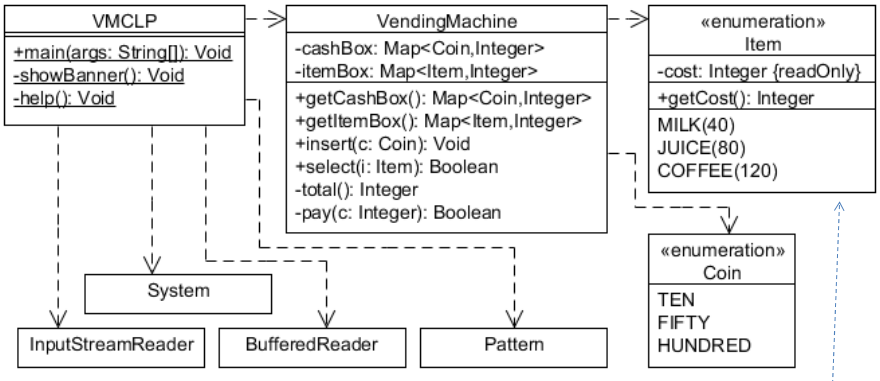
Sequence diagram (4)

loop; the fragment is repeatedly executed while the condition (#10YenCoin < c) holds

✓ 10*c is the cost of the item.
✓ if the number of 10Yen coins is greater than or equal to c, we can pay for the item.
✓ Otherwise, if there is a 50Yen (or 100Yen) coin, it is changed into five 10Yen coins (or five 10Yen & one 50Yen coins).
✓ Otherwise, we cannot.



Class diagram of vending machine



enumeration (enum type) in Java is a class and then it may have attributes (fields) and methods; 40 of MILK(40) is the attribute (cost) owned by the instance (object) MILK of the class

Vending machine in Java (1)

```
import java.util.*;
import java.io.*;
import java.util.regex.*;

public class VMCLP {
  private static void showBanner() {…}
  private static void help() {…}
  public static void main(String[] args) throws IOException {
    VendingMachine vm = new VendingMachine();
    String line;
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    showBanner();
    while ((line = br.readLine()) != null) {…}  }
}
```

package in which HashMap, ect. are declared

package in which BufferedReader, etc. are declared

package in which Pattern, etc. are declared

it decodes bytes read from the standard input into characters

it reads character from the standard input in a buffered way

it reads a line of text and returns the line excluding a line break as a String

will be explained in the following pages

Vending machine in Java (2)

```
line = line.trim();
if (line.equals("")) { // Nothing is done. }
else if (line.equals("stop")) {
  Map<Item,Integer> items = vm.getItemBox();
  Map<Coin,Integer> coins = vm.getCashBox();
  System.out.println("Items bought: " + items);
  System.out.println("Change: " + coins);
  System.out.println("bye bye!!");  break; }
else if (line.equals("help")) { help(); }
else if (line.equals("insert")) {
  System.err.println("Usage: insert [10|50|100]");  }
else if (line.equals("select")) {
  System.err.println("Usage: select [milk|juice|coffee]"); }
else if (line.equals("showItems")) { System.out.println(vm.getItemBox()); }
else if (line.equals("showCoins")) { System.out.println(vm.getCashBox()); }
```

it removes white spaces at both sides of the String

"Stop Buying" use case

"Show Usage" use case

"insert" needs an argument

"select" needs an argument

"Show Items" & "Show Coins" use cases

Vending machine in Java (3)

checking if line starts with “insert”, followed by one space or tab and then one or more characters

split into two strings such that the 1st string is the one just before the 1st space or tab, and the 2nd string is the one just after the 1st space or tab

```
else if (Pattern.matches("insert[ ¥t].*",line)) {
  String[] la = line.split("[ ¥t]",2);
  if (la.length < 2) {  // You never get here!
    assert false : "In insert command: " + line; }
  else { String a = la[1].trim();
    if (a.equals("10")) { vm.insert(Coin.TEN); }
    else if (a.equals("50")) { vm.insert(Coin.FIFTY); }
    else if (a.equals("100")) { vm.insert(Coin.HUNDRED); }
    else { System.err.println("Usage: insert [10|50|100]"); }  }  }
```

will be explained later

“Insert Coin” use case

Vending machine in Java (4)

“Select Item” use case

```
else if (Pattern.matches("select[ ¥t].*",line)) {
  String[] la = line.split("[ ¥t]",2);
  if (la.length < 2) {…} else { String a = la[1].trim();
    if (a.equals("milk")) {
      if (vm.select(Item.MILK)) { System.out.println("you bought one milk.");
      } else { System.out.println("Not enough coins!"); } }
    else if (a.equals("juice")) {
      if (vm.select(Item.JUICE)) { System.out.println("you bought one juice.");
      } else { System.out.println("Not enough coins!"); } }
    else if (a.equals("coffee")) {
      if (vm.select(Item.COFFEE)) {
        System.out.println("you bought one coffee.");
      } else { System.out.println("Not enough coins!"); } }
    else { System.err.println("Usage: select [milk|juice|coffee]"); }  }
  } else { System.err.println("No such a command!!!"); }
  System.out.print("VM> ");  }
```

Vending machine in Java (5)

assert can be used to check something that must hold.

```
public class TestAssert {
  public static void main(String[] args) {
    Integer m = new Integer(3);  Integer n = new Integer(3);
    assert m == n : "m & n refer to different objects";  }  }
```

Boolean expression

an expression such as a string (option)

"-ea" option such as "java -ea TestAssert" should be used to make assert effective. If so, this program throws an AssertionError exception because m == n becomes false.

If m & n are reference types, m == n becomes true if and only if m & n refer to exactly the same object; otherwise, it becomes false.

m.equals(n) must be true in the above example.

checks if the contents are the same

Vending machine in Java (6)

40, 80 & 120 are attributes of MILK, JUICE & COFFEE, respectively

declaration of constants should ended with ; if there are attributes/methods in an enum type

```
public enum Item {
   MILK(40), JUICE(80), COFFEE(120);
   private final int cost;
   private Item(int c) { cost = c; }
   public int getCost() { return cost; }
}
```

Constructors of enum types are private because any more instances cannot be made

attributes (fields) are final and then the attribute cost of each item cannot be changed

Vending machine in Java (7)

```
public class VendingMachine {
  private Map<Coin,Integer> cashBox;
  private Map<Item,Integer> itemBox;
  public VendingMachine() {
    cashBox = new HashMap<Coin,Integer>(); cashBox.put(Coin.TEN,0);
    cashBox.put(Coin.FIFTY,0); cashBox.put(Coin.HUNDRED,0);
    itemBox = new HashMap<Item,Integer>(); itemBox.put(Item.MILK,0);
    itemBox.put(Item.JUICE,0); itemBox.put(Item.COFFEE,0); }
  public Map<Coin,Integer> getCashBox() { return cashBox; }
  public Map<Item,Integer> getItemBox() { return itemBox; }
  public void insert(Coin c) { int x = cashBox.get(c); cashBox.put(c,x+1); }
  private int total() { return cashBox.get(Coin.TEN)*10
    +cashBox.get(Coin.FIFTY)*50+cashBox.get(Coin.HUNDRED)*100; }
  private boolean pay(int c) {…}
  public boolean select(Item i) {…} }
```

sequence diagram that starts with insert(c: Coin)

Vending machine in Java (8)

sequence diagram that starts with select(i: item)

```
public boolean select(Item i) {
    int cost = i.getCost();
    if (total() < cost) return false;
    boolean b = pay(cost/10);
    if (b) {
      int c = itemBox.get(i);
      itemBox.put(i,c+1);
      return true;
    } else {
      // you never get here!
      assert false : "cashBox: " + cashBox + ", " + "cost: " + cost;
      return false;
    }
  }
```

double check; there may be some flaws in the implementation of pay

Vending machine in Java (9)

sequence diagram that starts with pay(c: int)

```
private boolean pay(int c) {
  while (cashBox.get(Coin.TEN) < c) {
    if (cashBox.get(Coin.FIFTY) > 0) {
      int t1 = cashBox.get(Coin.FIFTY); int t2 = cashBox.get(Coin.TEN);
      cashBox.put(Coin.FIFTY,t1-1); cashBox.put(Coin.TEN,t2+5);
    } else if (cashBox.get(Coin.HUNDRED) > 0) {
      int t1 = cashBox.get(Coin.HUNDRED);
      int t2 = cashBox.get(Coin.FIFTY); int t3 = cashBox.get(Coin.TEN);
      cashBox.put(Coin.HUNDRED,t1-1);
      cashBox.put(Coin.FIFTY,t2+1); cashBox.put(Coin.TEN,t3+5);
    } else { return false; } }
    int t = cashBox.get(Coin.TEN);
    assert t >= c : "t >= c must hold but, " + "t=" + t + " & c=" + c;
    cashBox.put(Coin.TEN,t-c);
    return true;  }
```

50Yen coins is changed into five 10Yes coins

not enough cash

100Yen coins is changed into five 10Yen coins & one 50Yen coin

subtract c 10Yen coins from the cash box

Summary

- Use case
- Use case diagram
- State (machine) diagram
- Sequence diagram
- Class diagram of vending machine
- Vending machine in Java