Java與UML

第一章

UML概論

本章重點

- Uniform Modeling Language
- 行為圖型
- 架構圖型
- 動態圖型

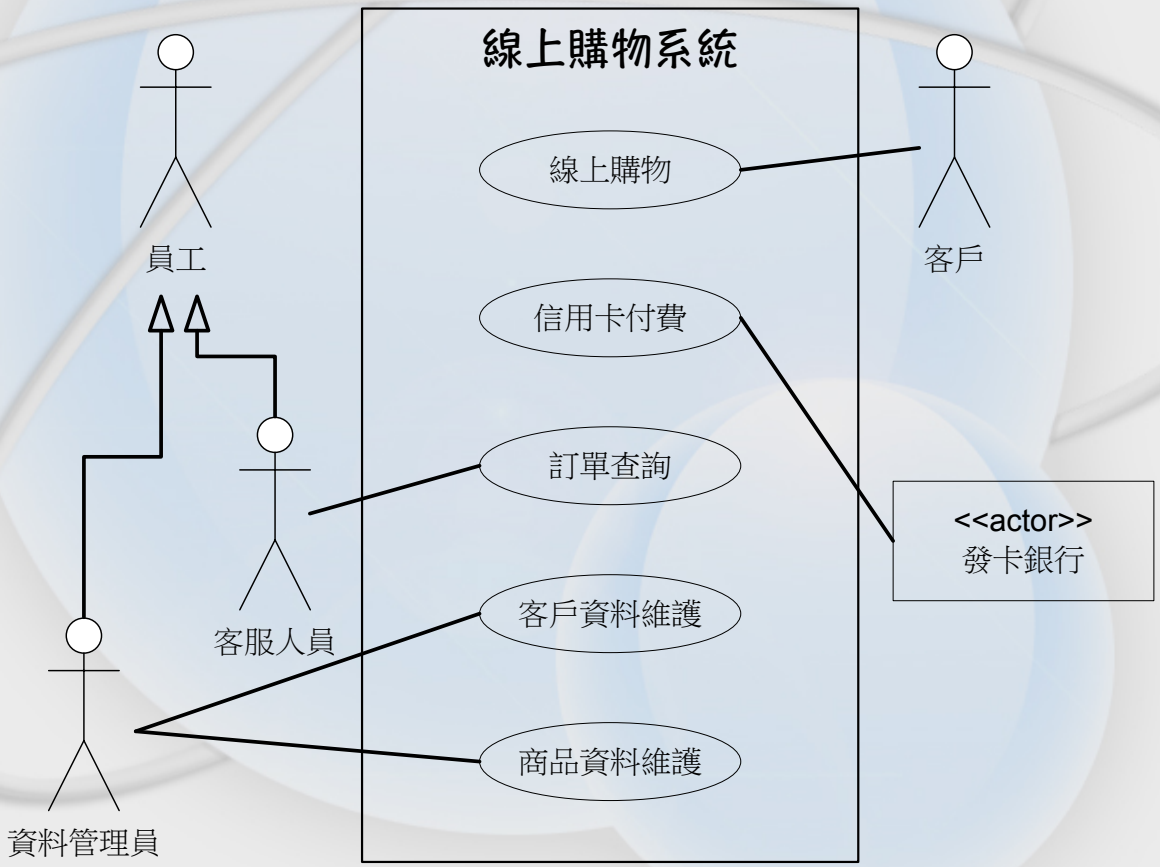
Uniform Modeling Language

- 使用圖型化的表示方式，用來表示軟體系統的圖型
- 與軟體開發行程無關

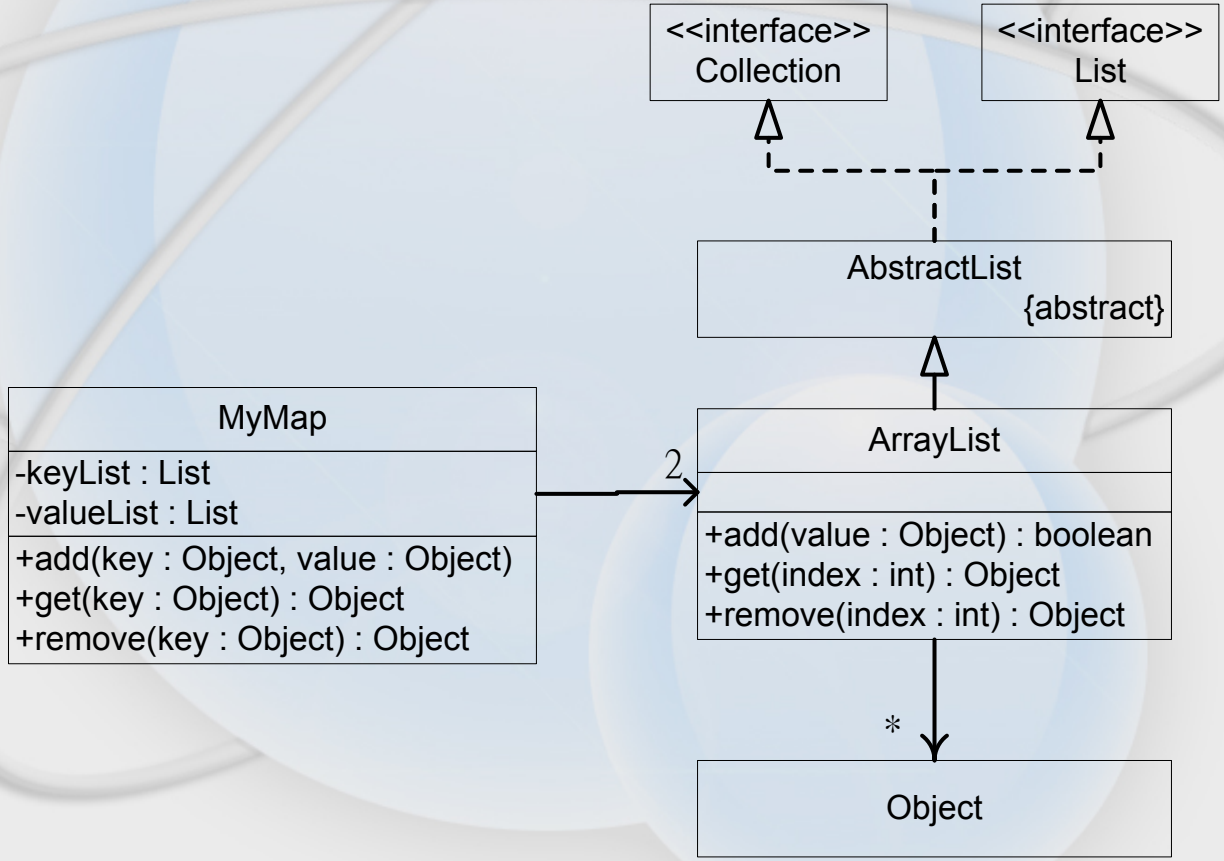
九種主要的圖型

- Use Case Diagrams
- Class Diagrams
- Object Diagrams
- Component Diagrams
- Deployment Diagrams
- Collaboration Diagrams
- Activity Diagrams
- Sequence Diagrams
- Statechart Diagrams

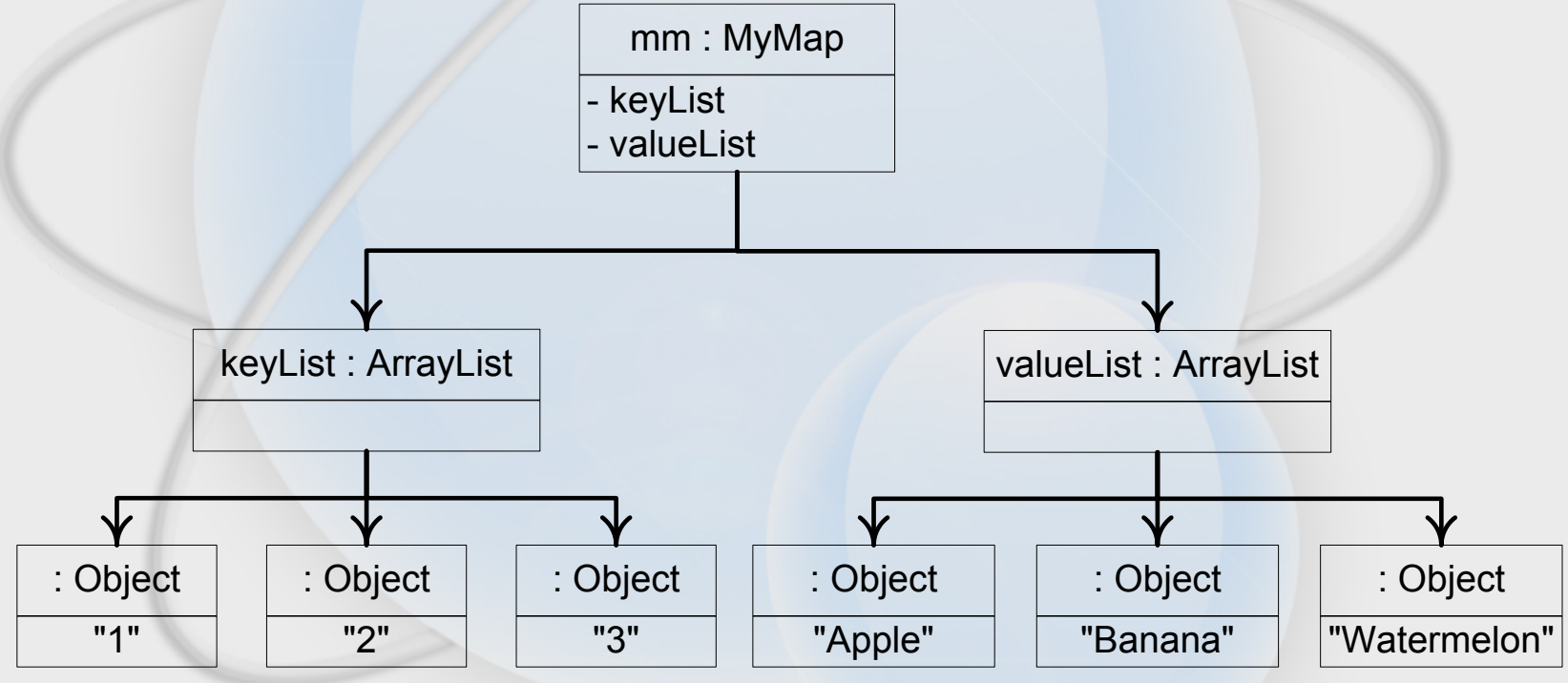
使用案例圖型


線上購物系統
線上購物
信用卡付費
訂單查詢
客戶資料維護
商品資料維護
員工
客戶
客服人員
資料管理員
<<actor>>
發卡銀行

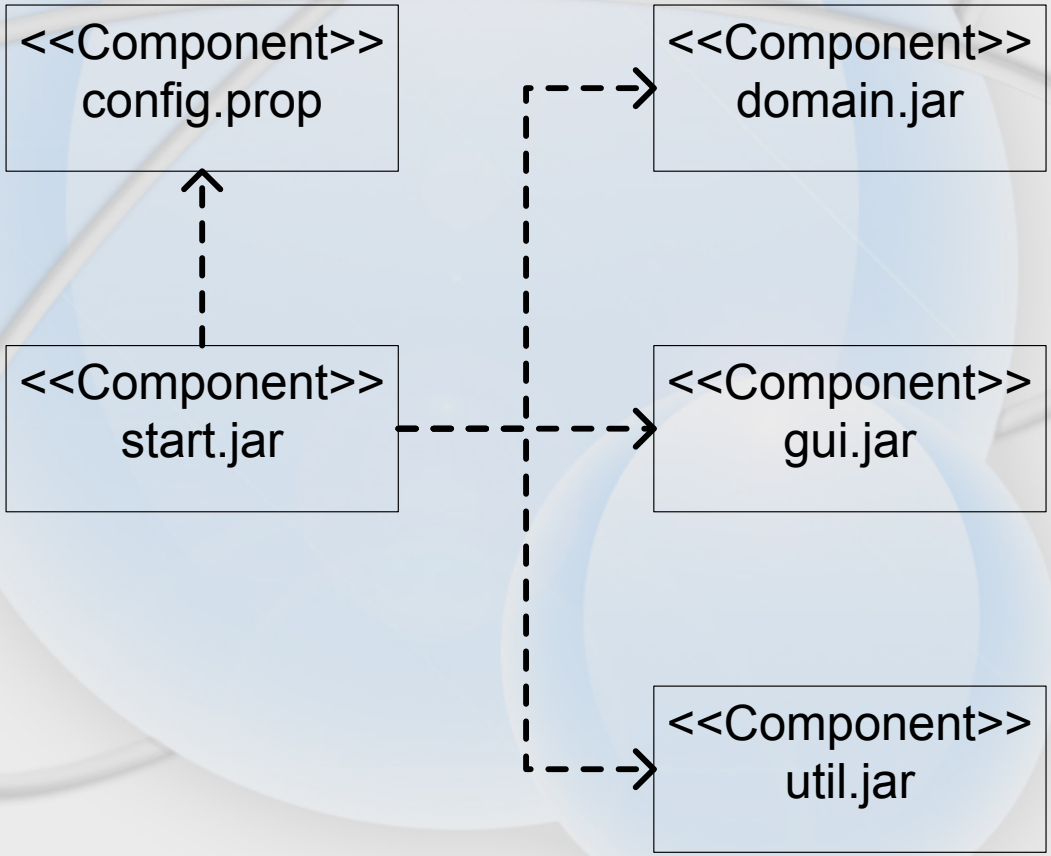
類別圖型


<<interface>>
Collection
<<interface>>
List
AbstractList
{abstract}
MyMap
-keyList : List
-valueList : List
+add(key : Object, value : Object)
+get(key : Object) : Object
+remove(key : Object) : Object
2
ArrayList
+add(value : Object) : boolean
+get(index : int) : Object
+remove(index : int) : Object
*
Object

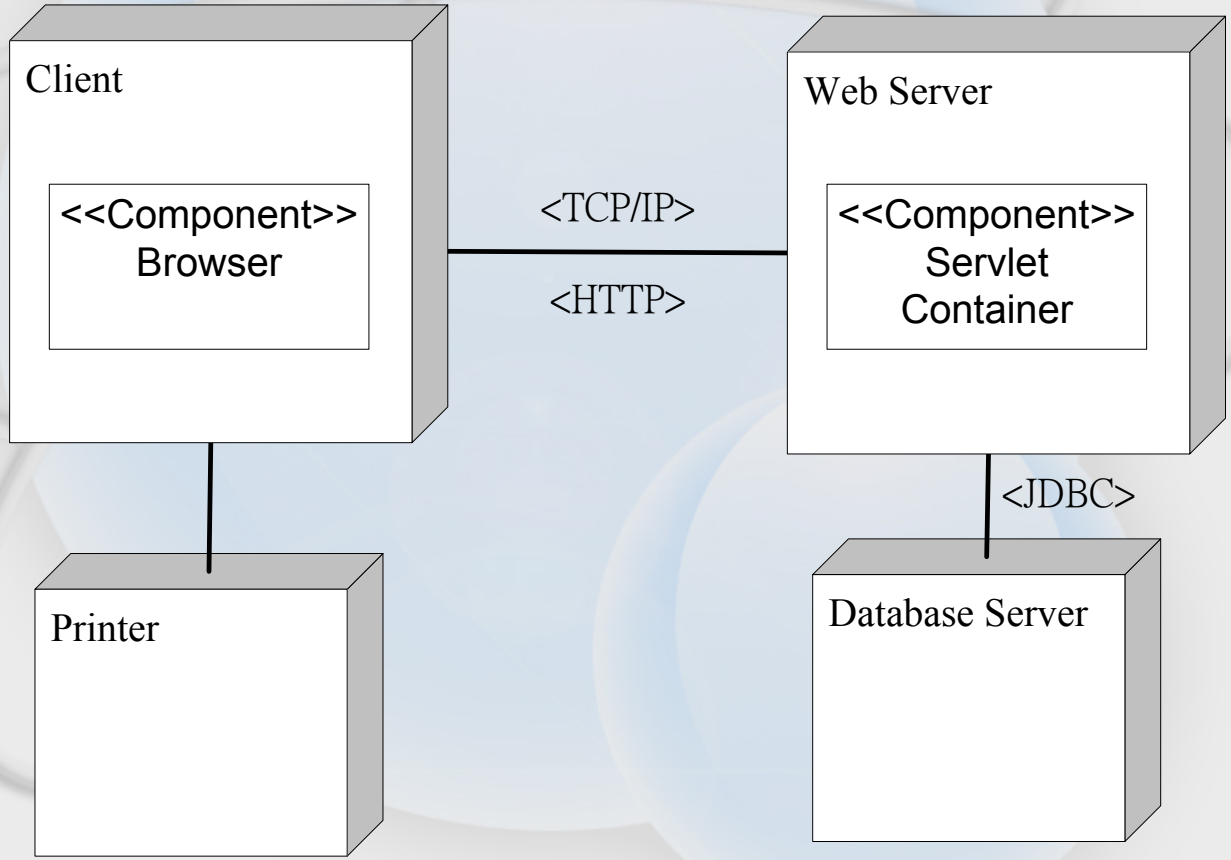
物件圖型


mm : MyMap
- keyList
- valueList
keyList : ArrayList
valueList : ArrayList
: Object
"1"
: Object
"2"
: Object
"3"
: Object
"Apple"
: Object
"Banana"
: Object
"Watermelon"

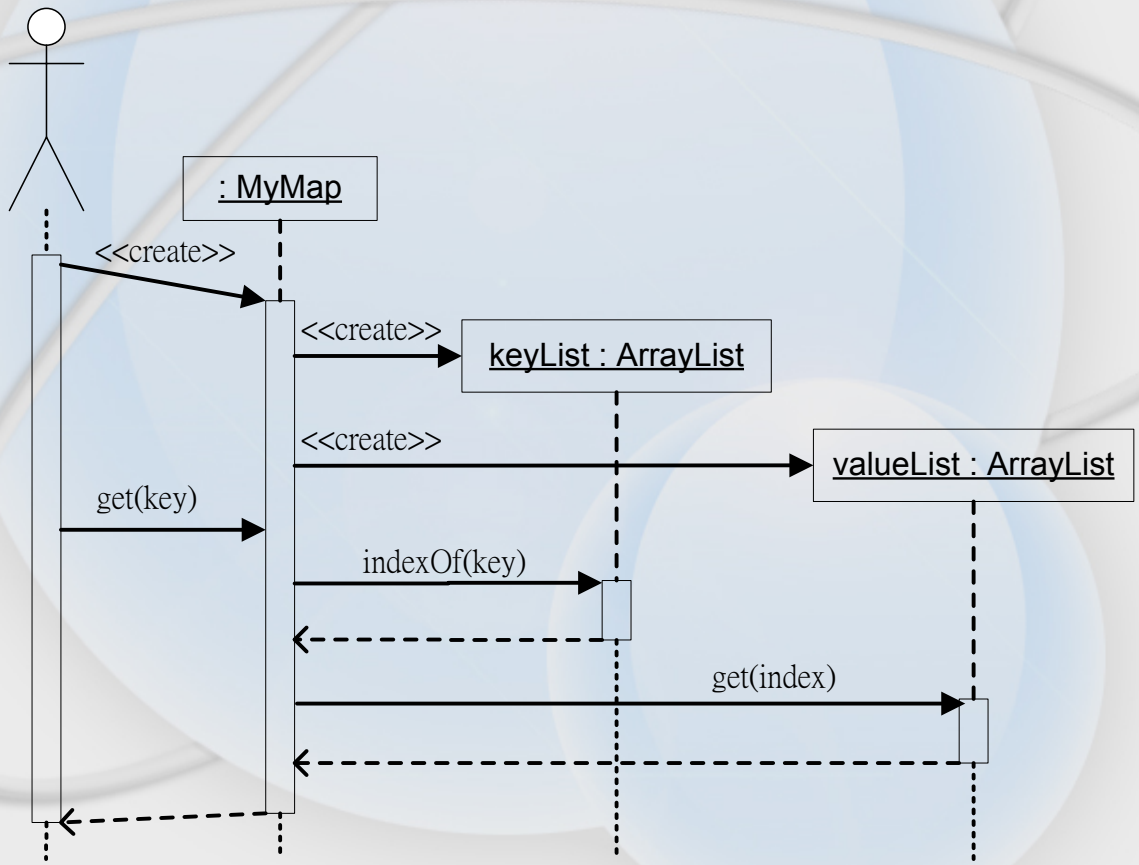
元件圖型


<<Component>>
config.prop
<<Component>>
domain.jar
<<Component>>
start.jar
<<Component>>
gui.jar
<<Component>>
util.jar

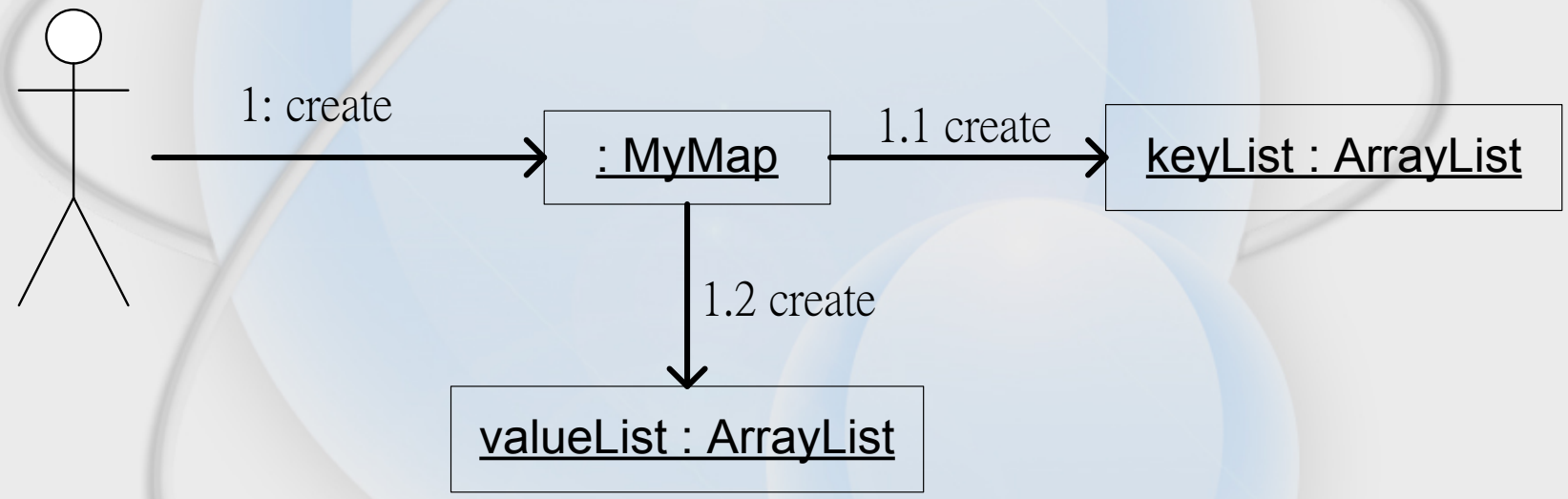
佈署圖型


Client
<<Component>>
Browser
<TCP/IP>
<HTTP>
Web Server
<<Component>>
Servlet
Container
<JDBC>
Printer
Database Server

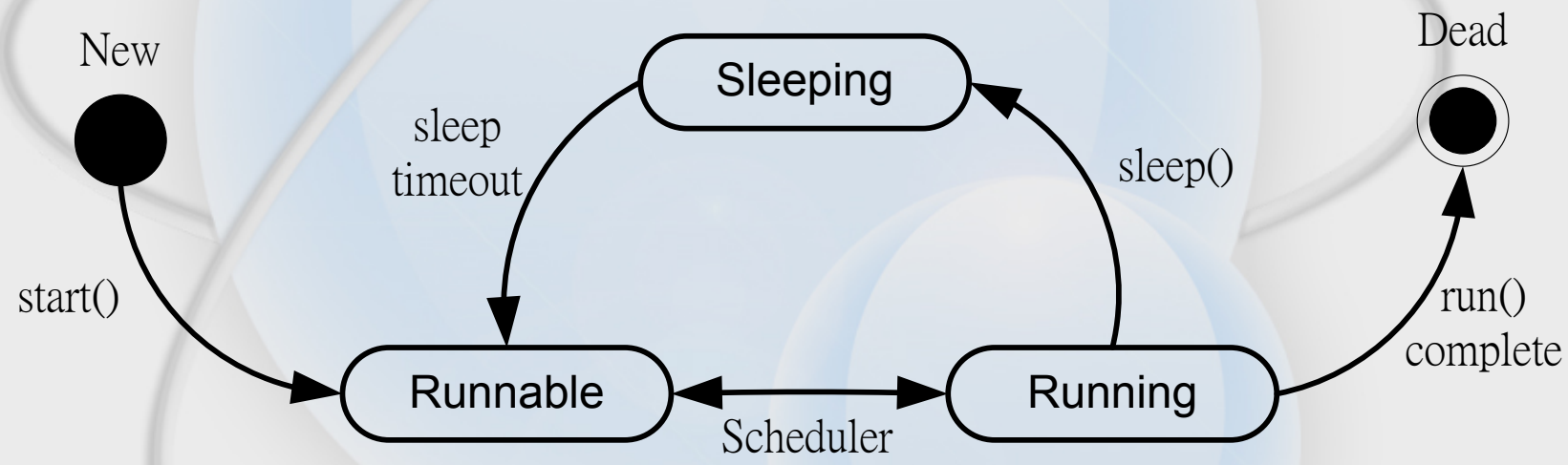
循序圖型


: MyMap
<<create>>
<<create>>
keyList : ArrayList
<<create>>
valueList : ArrayList
get(key)
indexOf(key)
get(index)

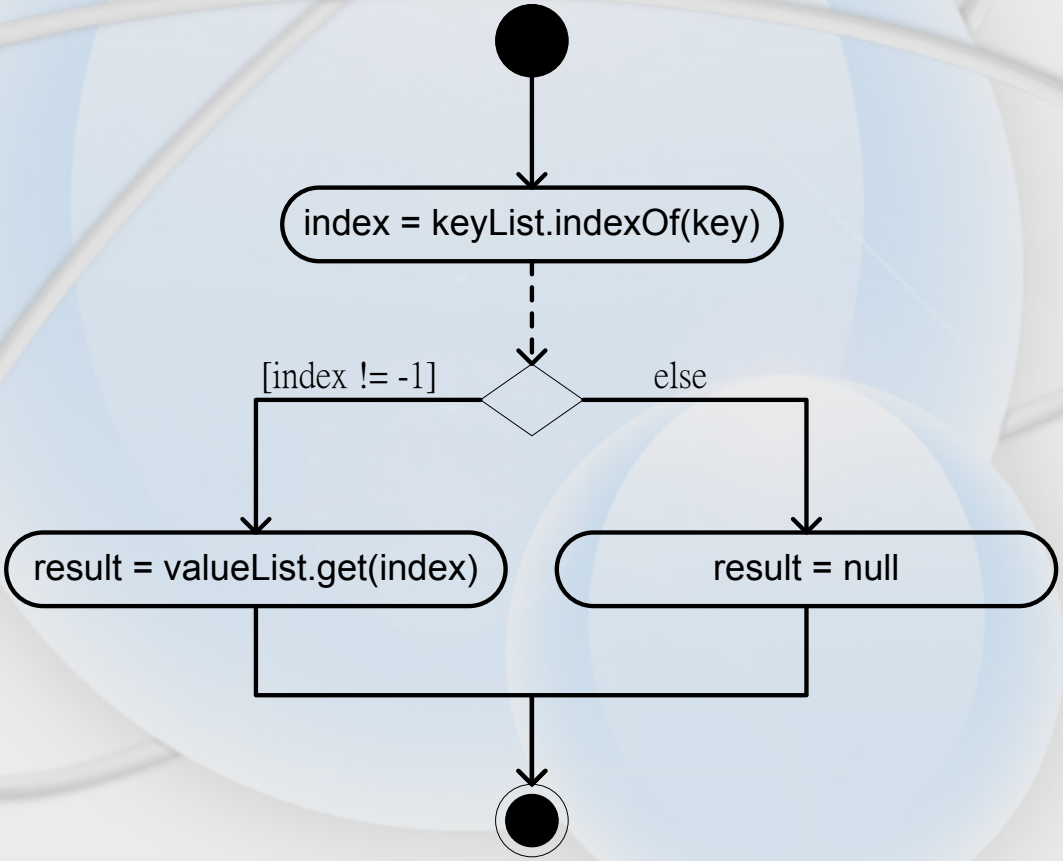
合作圖型


1: create
: MyMap
1.1 create
keyList : ArrayList
1.2 create
valueList : ArrayList

狀態圖型


New
Sleeping
Dead
sleep
timeout
sleep()
start()
run()
complete
Runnable
Running
Scheduler

活動圖型


index = keyList.indexOf(key)
[index != -1]
else
result = valueList.get(index)
result = null

第二章

類別圖型

本章重點

- 類別節點
- 結合關係

類別節點

- 類別(class)
- 抽象類別(abstract class)
- 介面(interface)

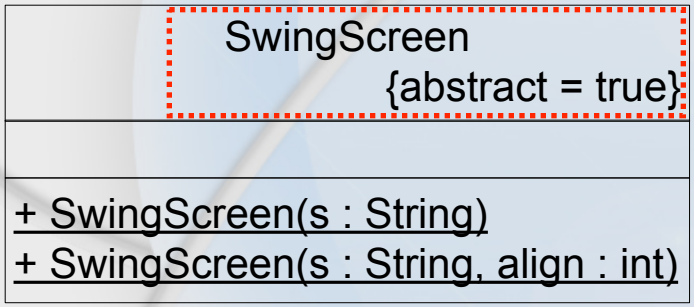
類別節點區格

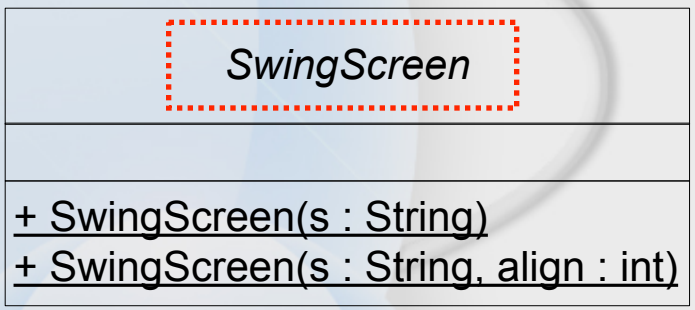
Calculator

- NO_OP : char = '\0'
- PLUS : char = '+'
- SUBTRACT : char = '-'
- MULTIPLY : char = '*'
- DIVIDE : char = '/'
- number1 : float = 0.0F
- operator : char = NO_OP

+ opEquals(number : String)
+ opAdd(number : String)
+ opSubtract(number : String)
+ opMultiply(number : String)
+ opDivide(number : String)
- performOperation(number2 : float) : float
parseNumber(number : String) : float

抽象類別


SwingScreen
{abstract = true}
+ SwingScreen(s : String)
+ SwingScreen(s : String, align : int)

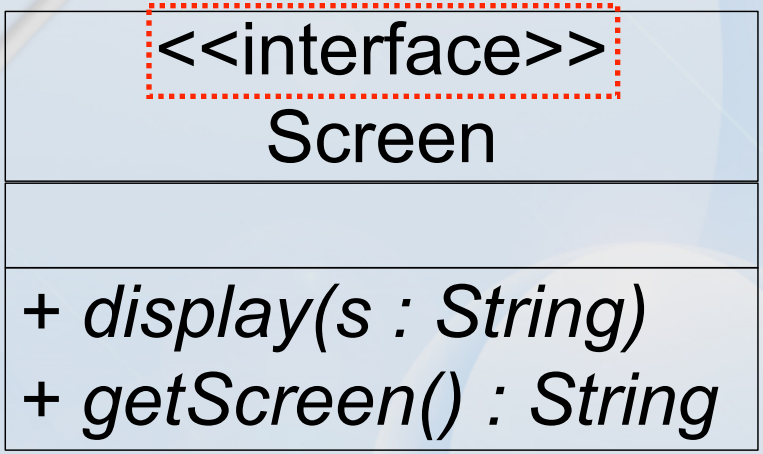

SwingScreen
+ SwingScreen(s : String)
+ SwingScreen(s : String, align : int)

額外的資訊

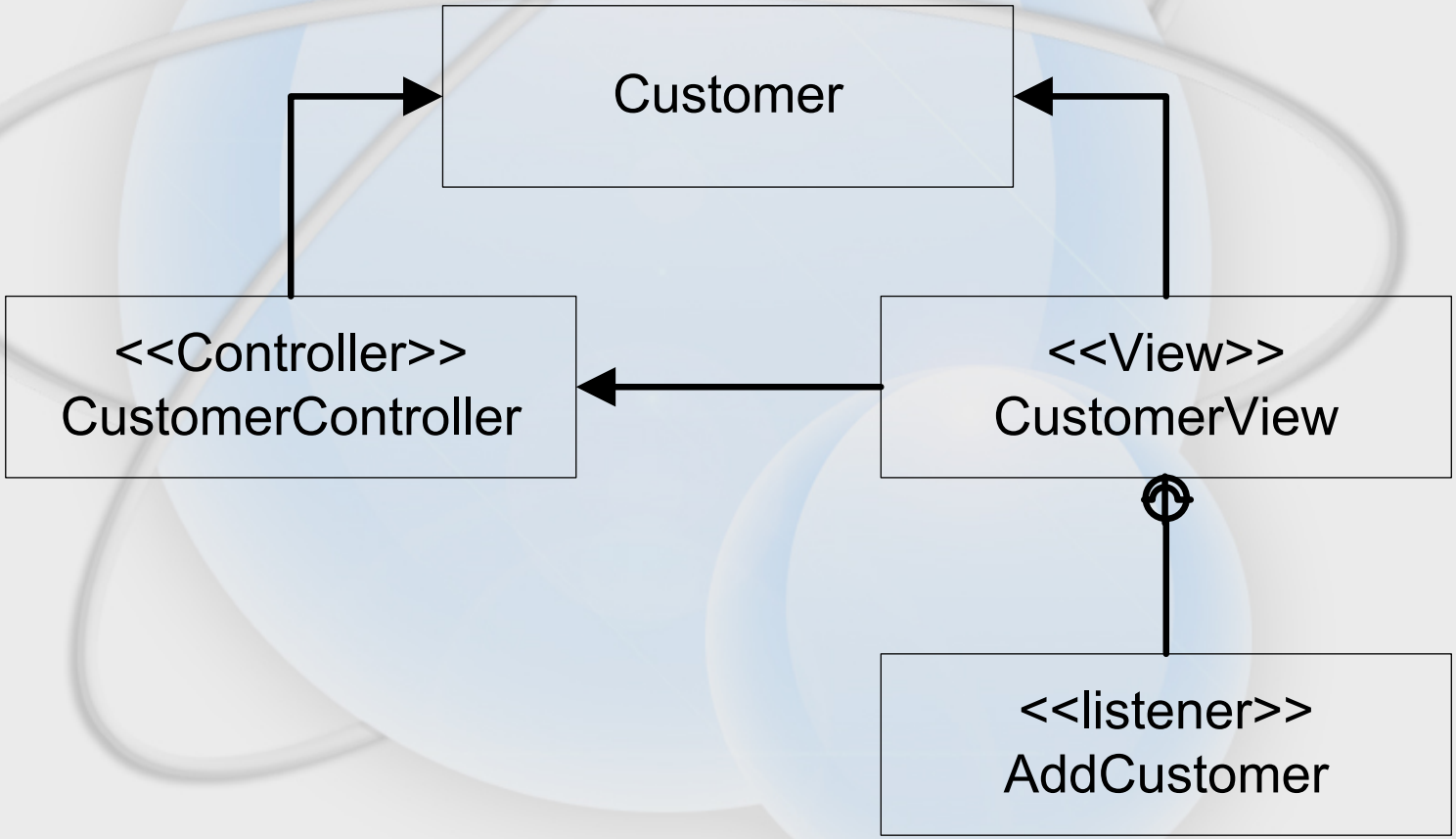
CalculatorScreen

{author = Simon,
file = CalculatorScreen.java,
date = 20010120,
public}

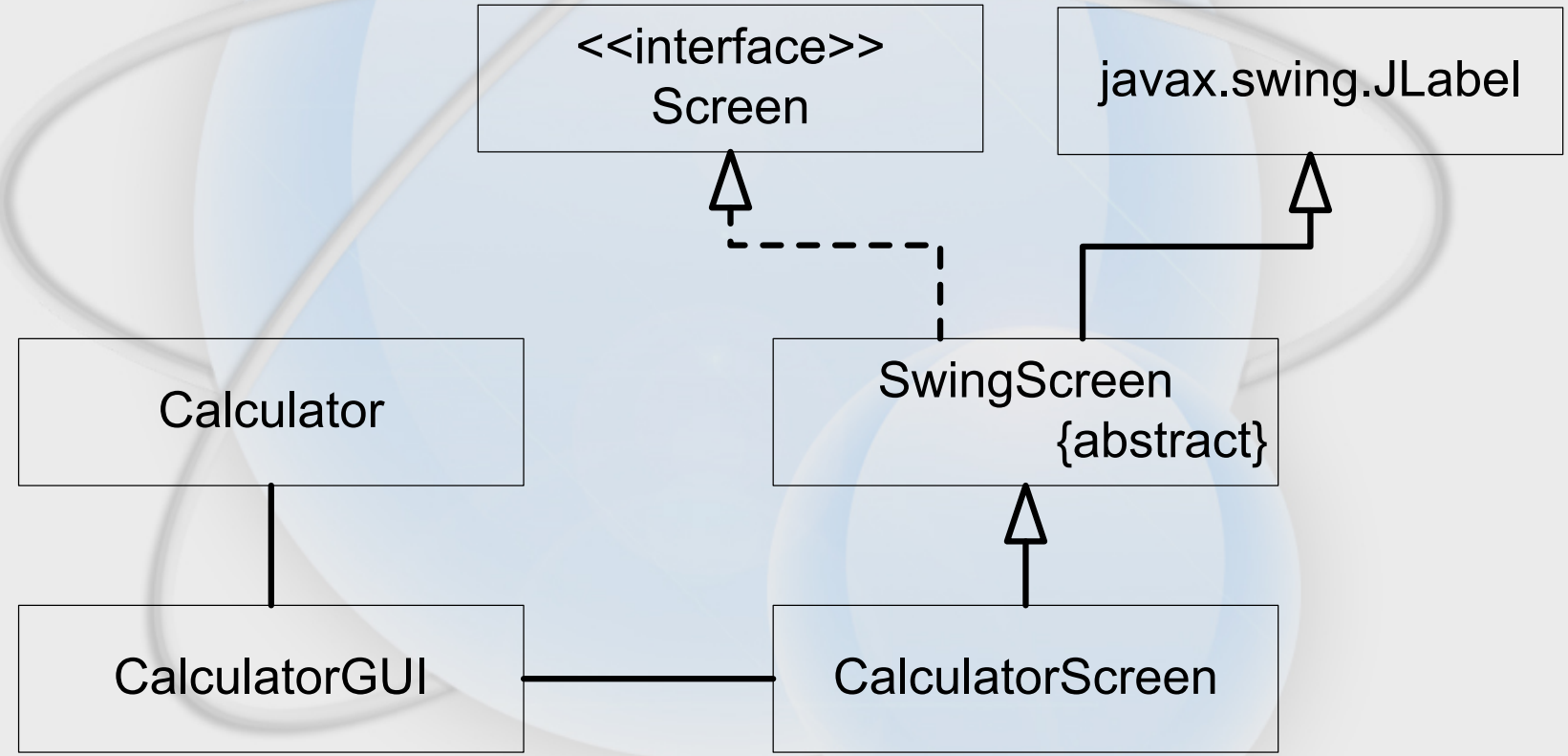
介面



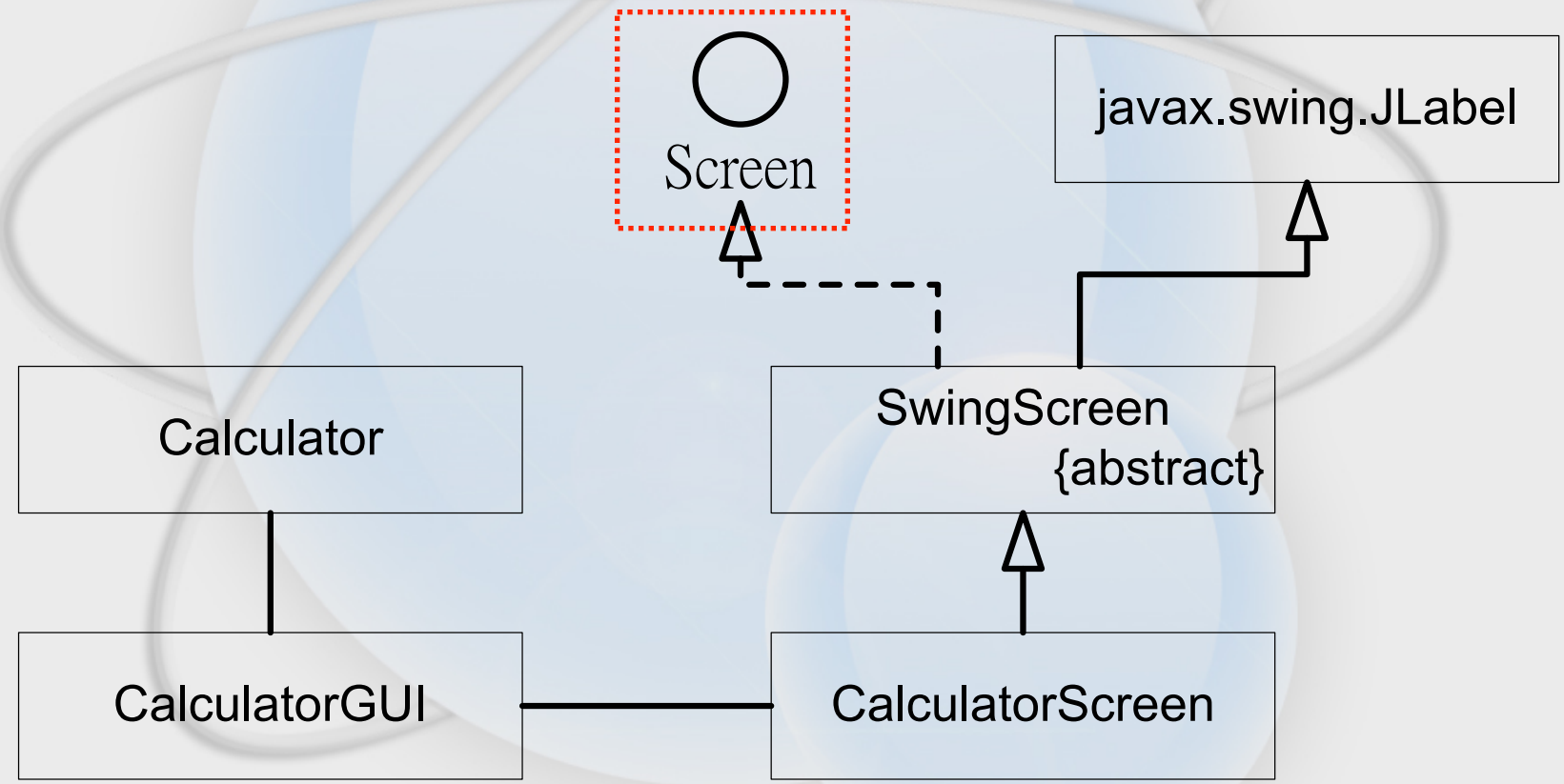
Stereotypes



使用名稱區格


<<interface>>
Screen
javax.swing.JLabel
Calculator
SwingScreen
{abstract}
CalculatorGUI
CalculatorScreen

介面表示方法


Screen
javax.swing.JLabel
Calculator
SwingScreen
{abstract}
CalculatorGUI
CalculatorScreen

屬性區格宣告語法

<存取範圍> <屬性名稱> [[<數量>] [<順序性>]] : <型態> [= <初始值>]

存取範圍修飾子	符號
public	+
protected	#
package private	~
private	-

屬性區格宣告語法

<存取範圍> <屬性名稱> [[<數量>] [<順序性>]] : <型態> [= <初始值>]

表示法	說明
數字	確定的數量
*	零到多個
0..*	零到多個
0..1	零到一個
1..*	一到多個
n..m	最少n個，最多m個

屬性區格宣告語法

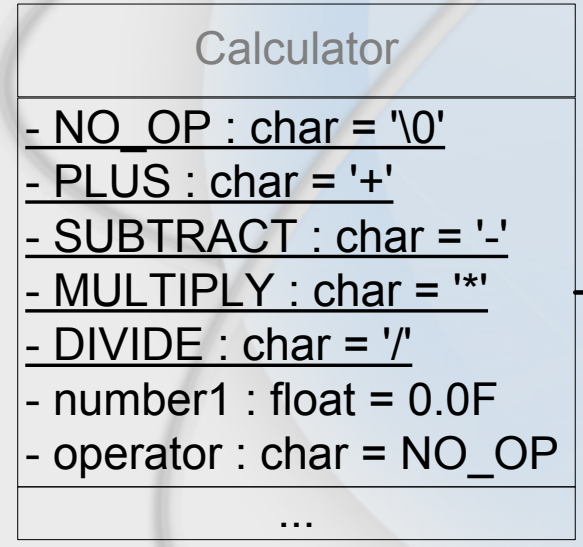
<存取範圍> <屬性名稱> [[<數量>] [<順序性>]] : <型態> [= <初始值>]

關鍵字	說明
unordered	不需要排序
ordered	需要排序

屬性宣告資訊

Calculator
- NO_OP : char = '\0' - PLUS : char = '+' - SUBTRACT : char = '-' - MULTIPLY : char = '*' - DIVIDE : char = '/' - number1 : float = 0.0F - operator : char = NO_OP
...

屬性宣告資訊



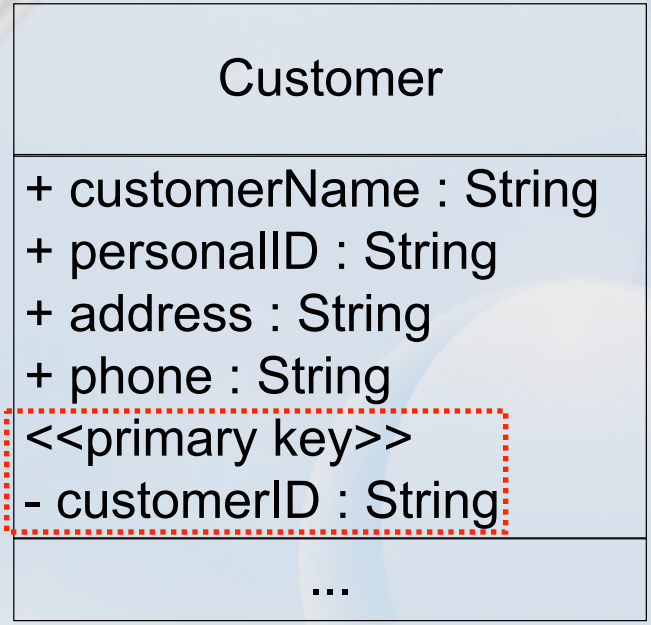

```
private static final char NO_OP = '\0';
private static final char PLUS = '+';
private static final char SUBTRACT = '-';
private static final char MULTIPLY = '*';
private static final char DIVIDE = '/';
private float  number1 = 0.0F;
private char   operator = NO_OP;
```

數量表示方法

Employee
- emails [1..3 unordered] : String
...

```
public class Employee {

  private String emails[] = new String[3];

}
```

使用Stereotypes



方法區格宣告語法

<存取範圍> <方法名稱>（[<參數>]）：<回傳型態>

存取範圍修飾子	符號
public	+
protected	#
package private	~
private	-

方法區格宣告語法

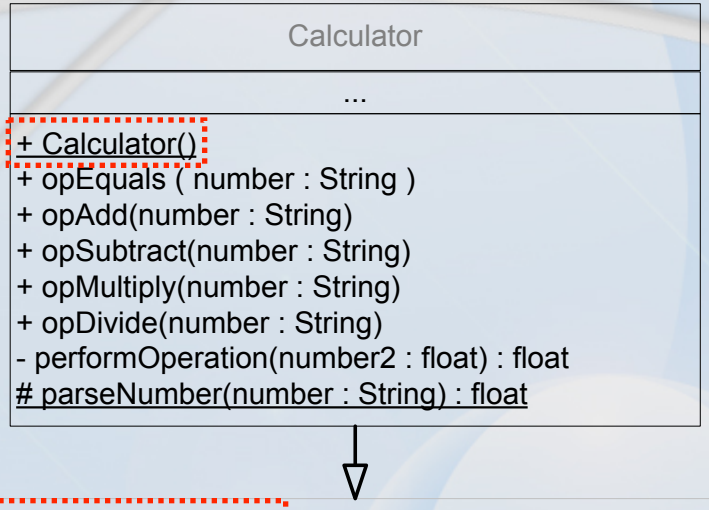
<存取範圍> <方法名稱> ([<參數>]) : <回傳型態>

[種類] <參數名稱> : <參數型態>

方法宣告資訊

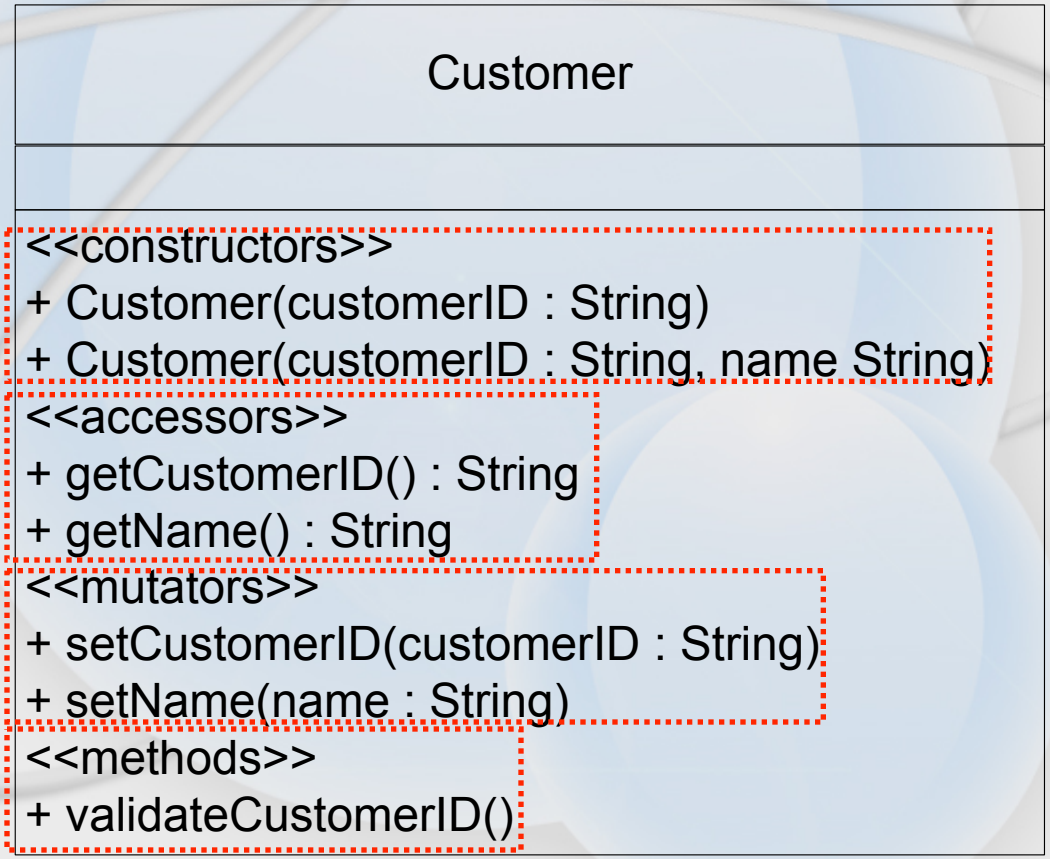
Calculator
...
+ opEquals (number : String)
+ opAdd(number : String)
+ opSubtract(number : String)
+ opMultiply(number : String)
+ opDivide(number : String)
- performOperation(number2 : float) : float
parseNumber(number : String) : float

方法宣告資訊




```
public Calculator() { ... }
public String opEquals(String number) { ... }
public String opAdd(String number) { ... }
public String opSubtract(String number) { ... }
public String opMultiply(String number) { ... }
public String opDivide(String number) { ... }
private float performOperation(float number2) { ... }
protected static float parseNumber(String number) { ... }
```

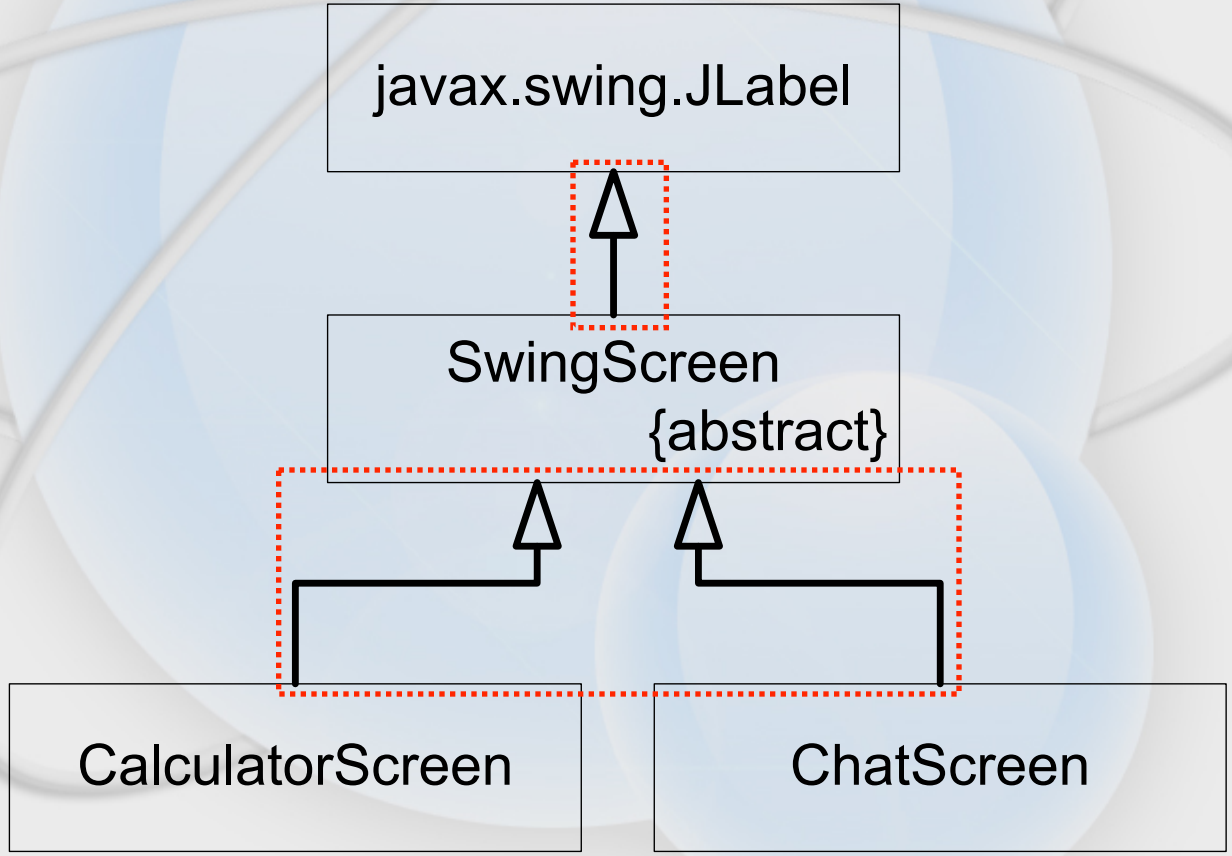
使用Stereotypes



抽象方法

java.util.Calendar {abstract}
... *# computeFields()* *# computeTime()* + set(field : int, value : int) + set(year : int, month : int, date : int) ...

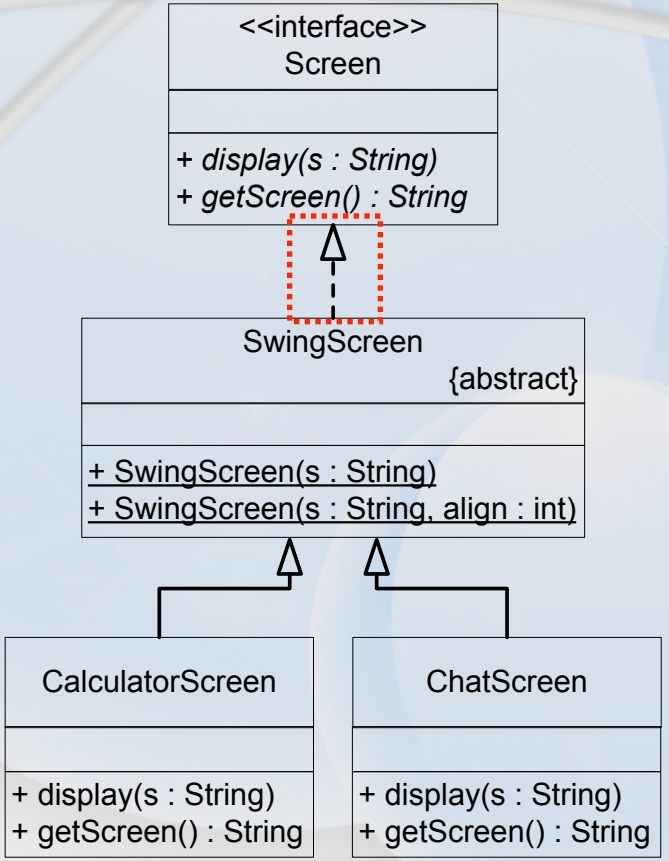
繼承


javax.swing.JLabel
SwingScreen
{abstract}
CalculatorScreen
ChatScreen

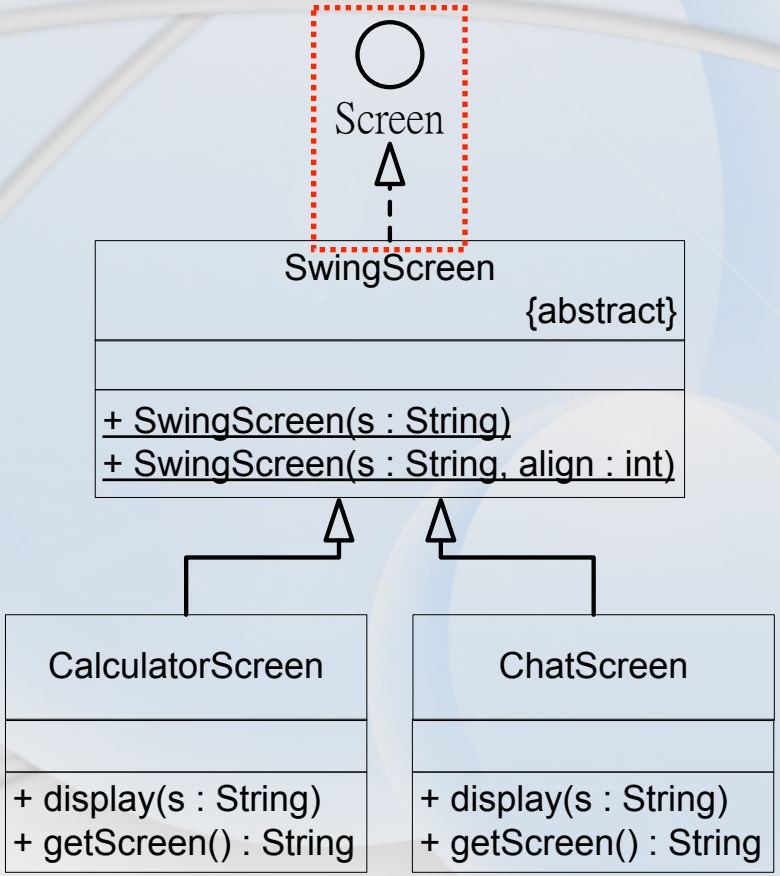
繼承

```
public abstract class SwingScreen extends javax.swing.JLabel {
}

public class CalculatorScreen extends SwingScreen {
}

public class ChatScreen extends SwingScreen {
}
```

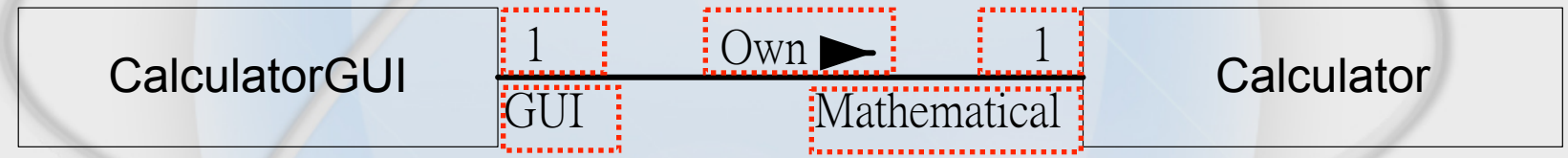
實作


<<interface>>
Screen
+ display(s : String)
+ getScreen() : String
SwingScreen
{abstract}
+ SwingScreen(s : String)
+ SwingScreen(s : String, align : int)
CalculatorScreen
+ display(s : String)
+ getScreen() : String
ChatScreen
+ display(s : String)
+ getScreen() : String

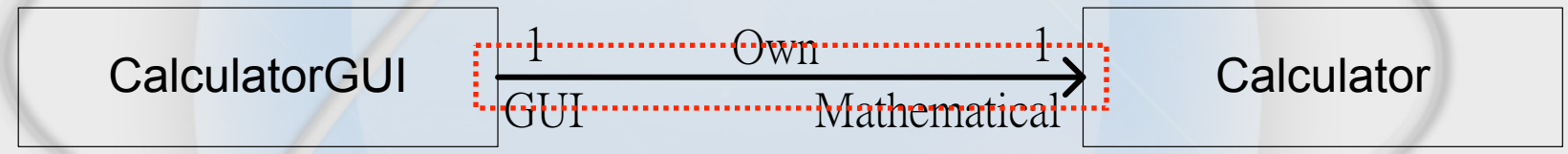
介面表示方法


Screen
SwingScreen
{abstract}
+ SwingScreen(s : String)
+ SwingScreen(s : String, align : int)
CalculatorScreen
+ display(s : String)
+ getScreen() : String
ChatScreen
+ display(s : String)
+ getScreen() : String

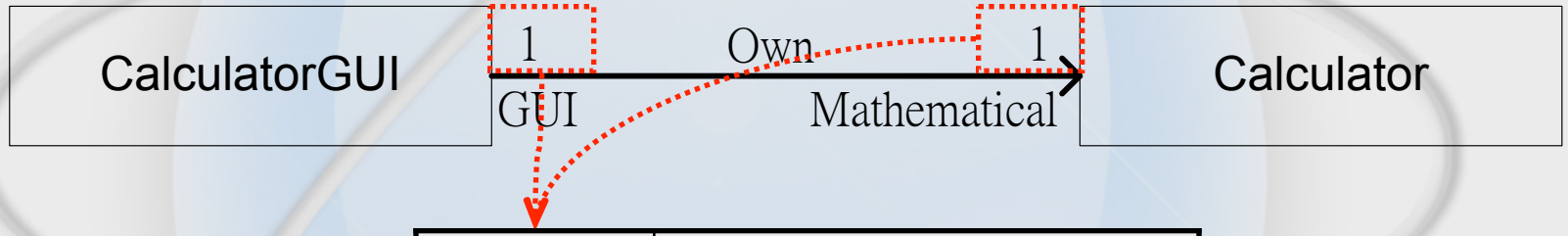
結合



結合表示方法

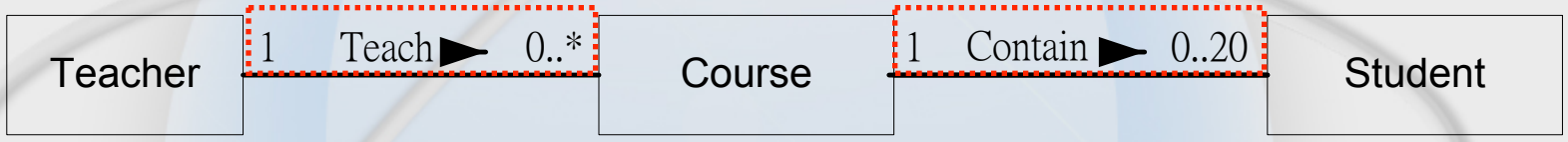


數量表示方法



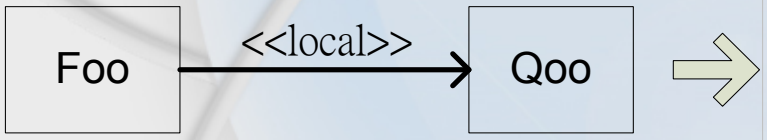
表示法	說明
數字	確定的數量
*	零到多個
0..*	零到多個
0..1	零到一個
1..*	一到多個
n..m	最少n個，最多m個

數量表示方法




```
public class Teacher {
 private Set classes = new Set();
}

public class Course {
 private Student students[] = new Student[20];
}
```

<<local>>

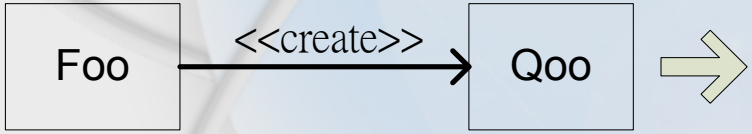



```
public class Foo {
  public void methodA() {
    Qoo q = new Qoo();
  }
}
```

<<parameter>>


```
public class Foo {
  public void methodA(Qoo q) {
    ...
  }
}
```

<<create>>

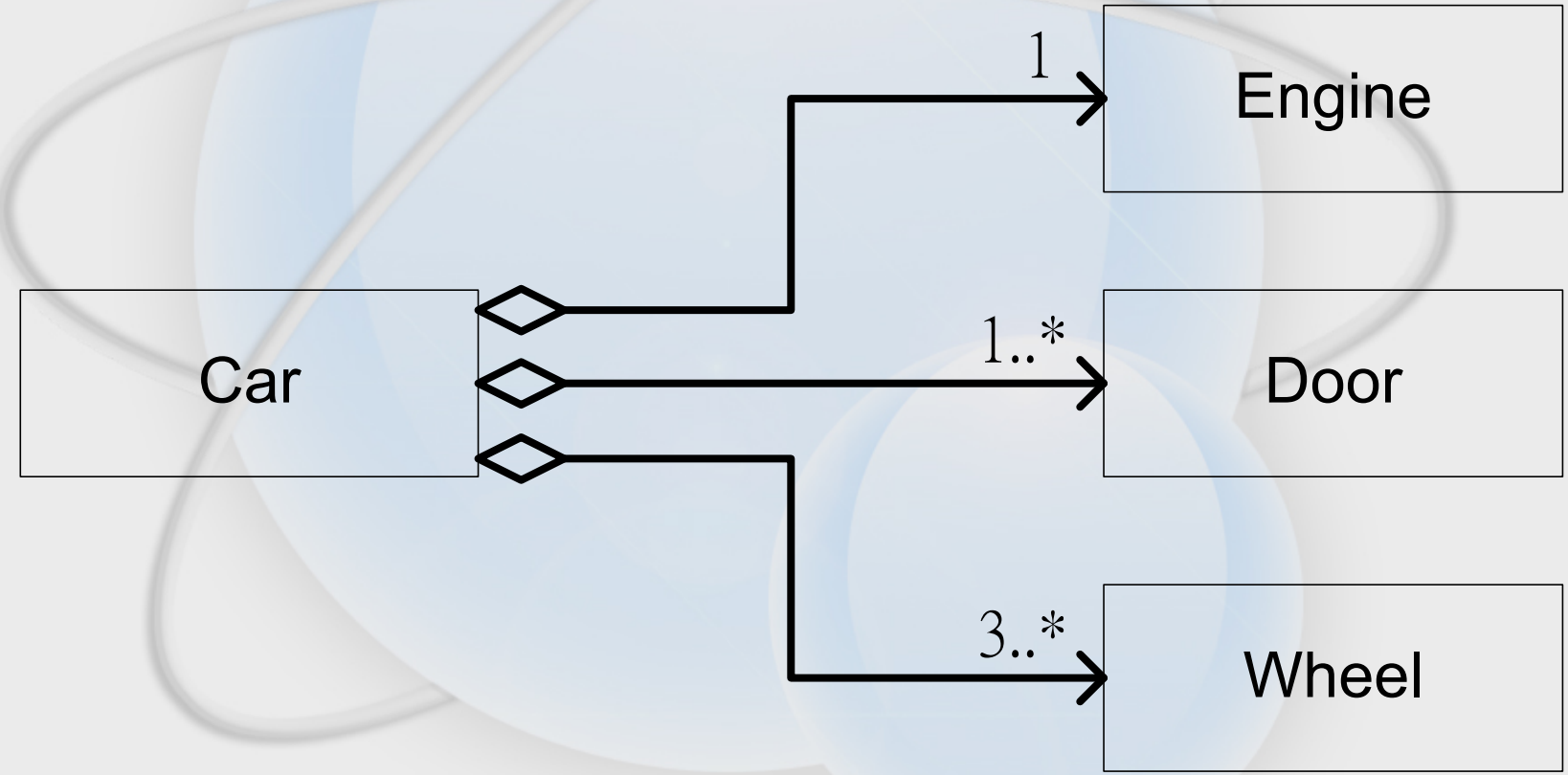



```
public class Foo {
  public Qoo getQoo() {
    return new Qoo();
  }
}
```

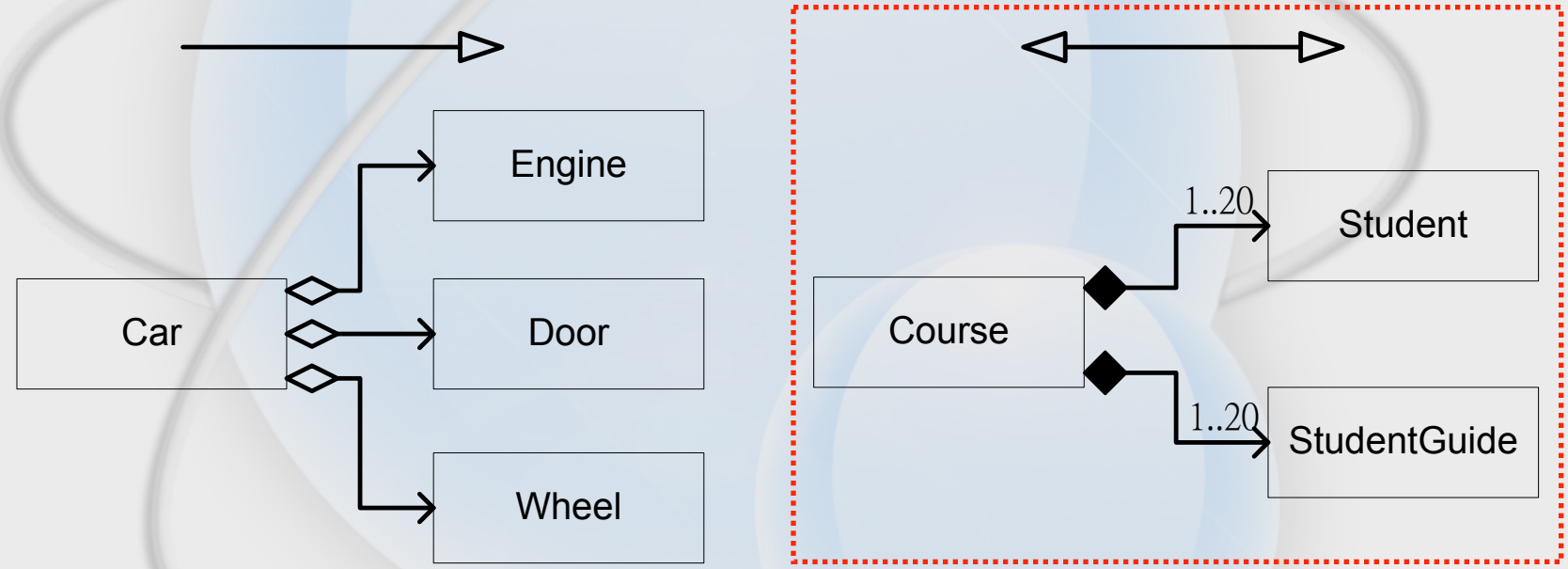
<<delegate>>


```
public class Foo {
  private Qoo q;
  public void methodA() {
    q.methodA();
  }
}
```

聚合



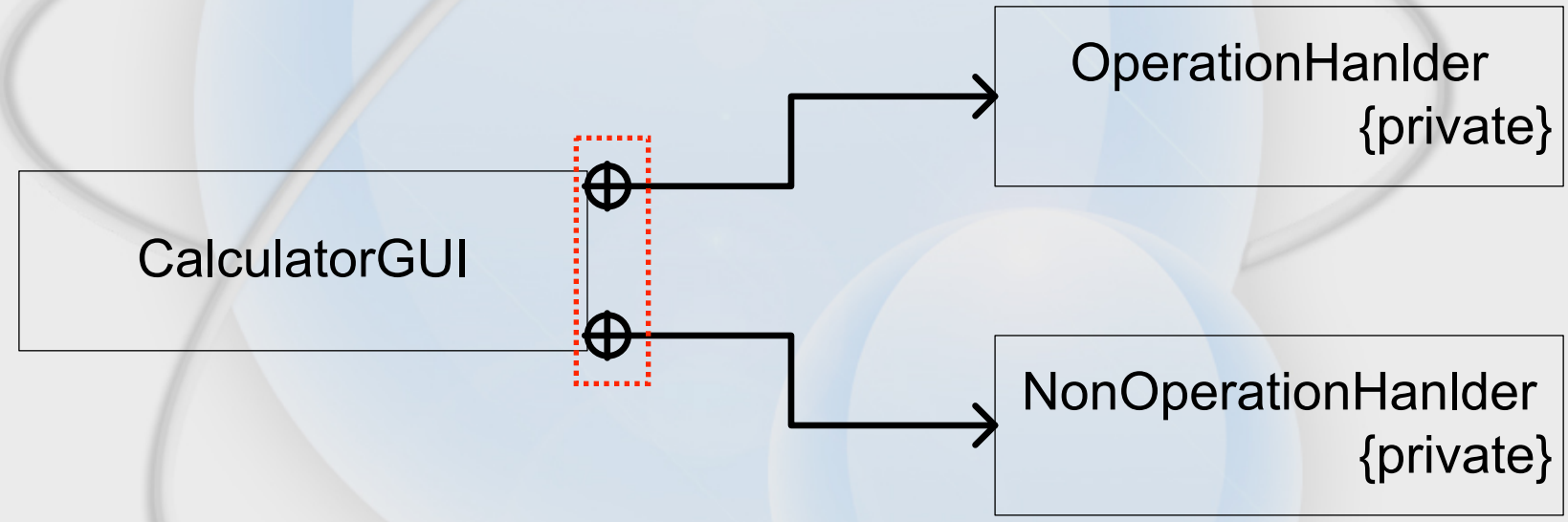
組合



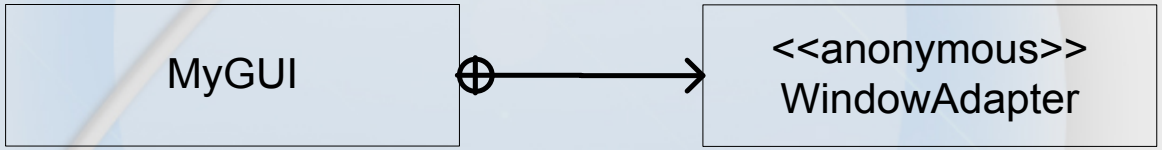
組合的特性

- 一個成員物件不能同時被兩個擁有者擁有
- 擁有者要負責成員的的生命週期

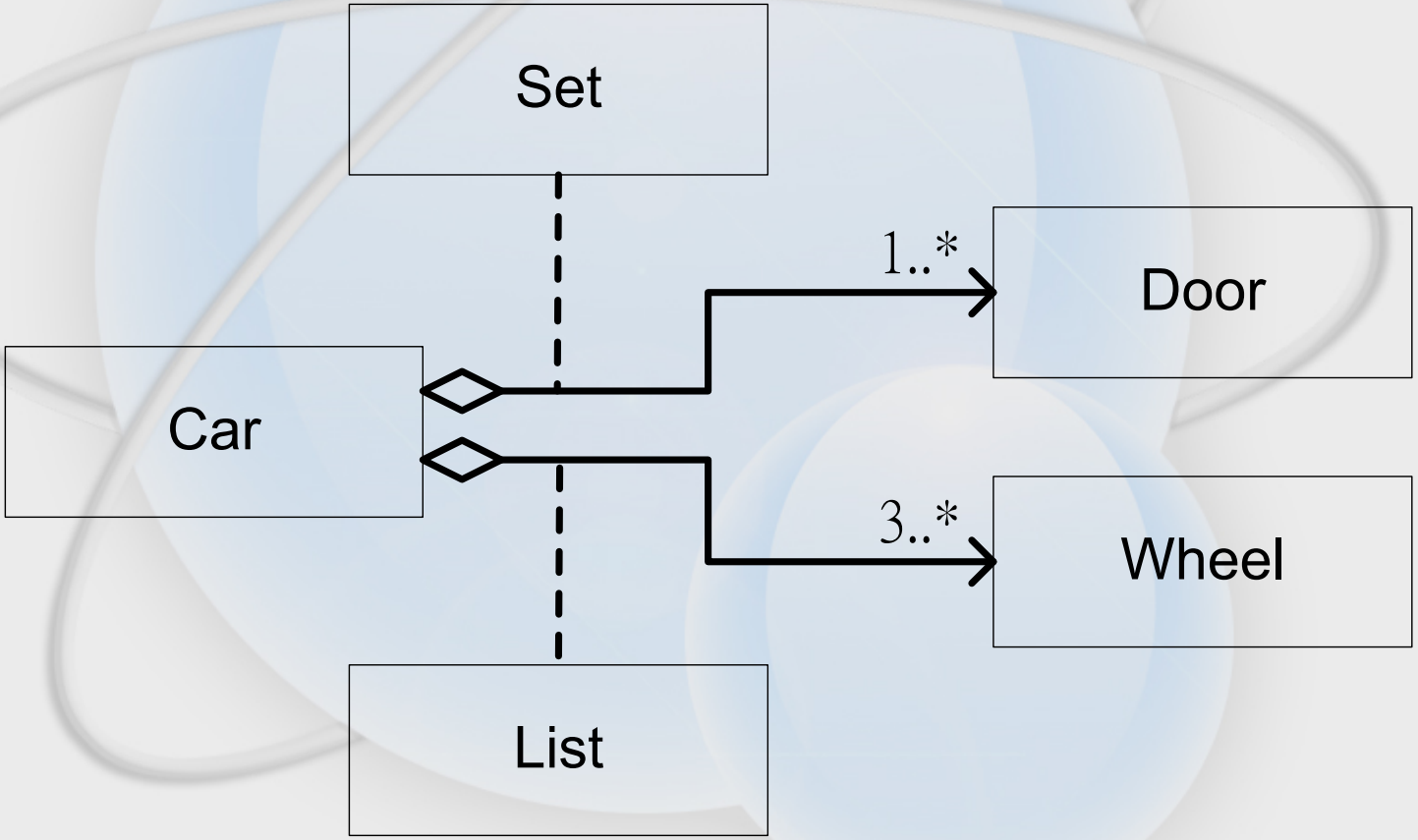
巢狀類別


CalculatorGUI
OperationHanlder
{private}
NonOperationHanlder
{private}

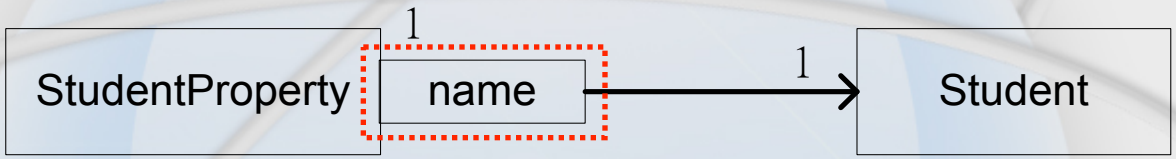
匿名類別



結合類別



限定結合




```
import java.util.*;
public class StudentProperty {
 private Map students = new HashMap();

  public Student getStuednt(String name) {
    return (Student) students.get(name);
  }

  public void addStudent(Student s) {
    students.put(s.getName(), s);
  }
}
```

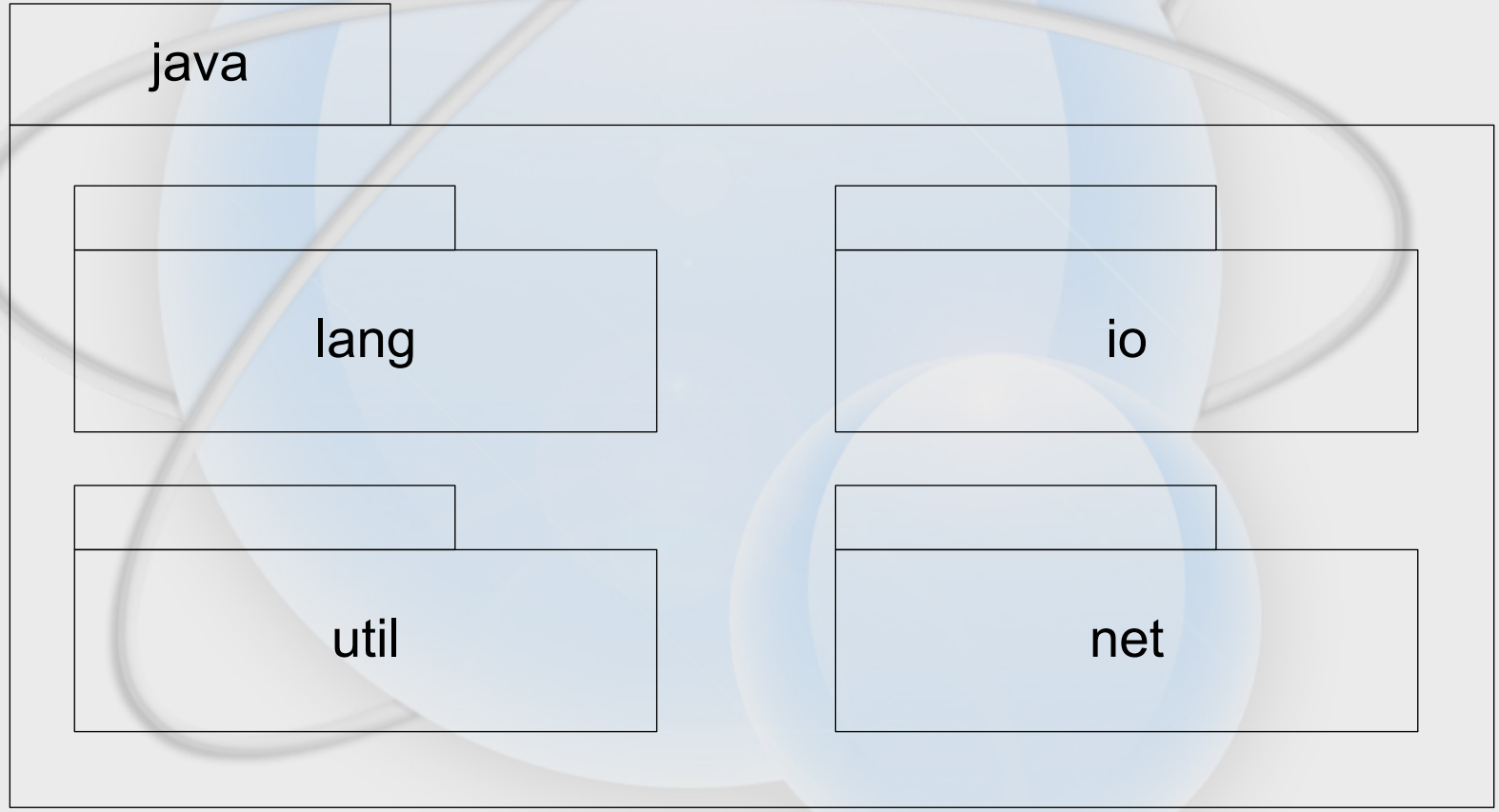
第三章

套件圖型

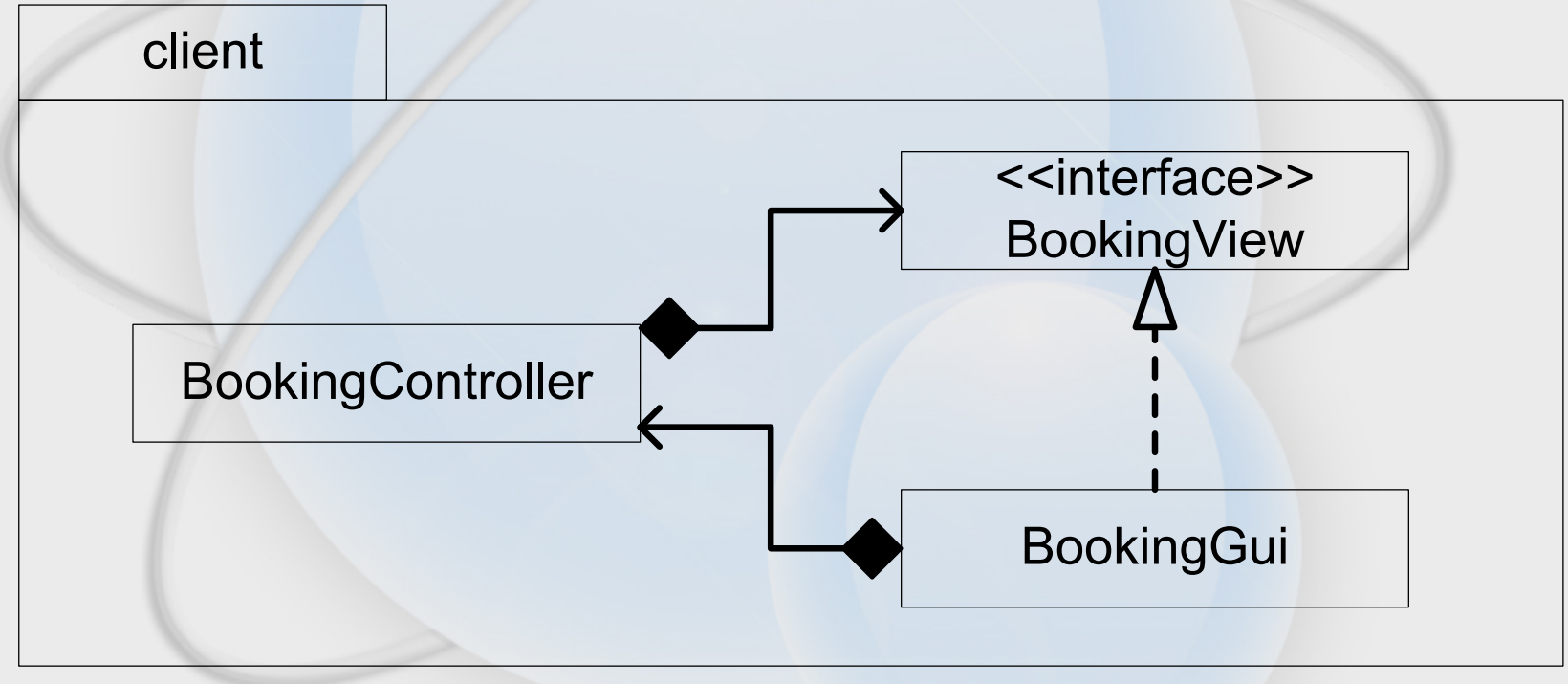
本章重點

- 套件標記
- 相依性

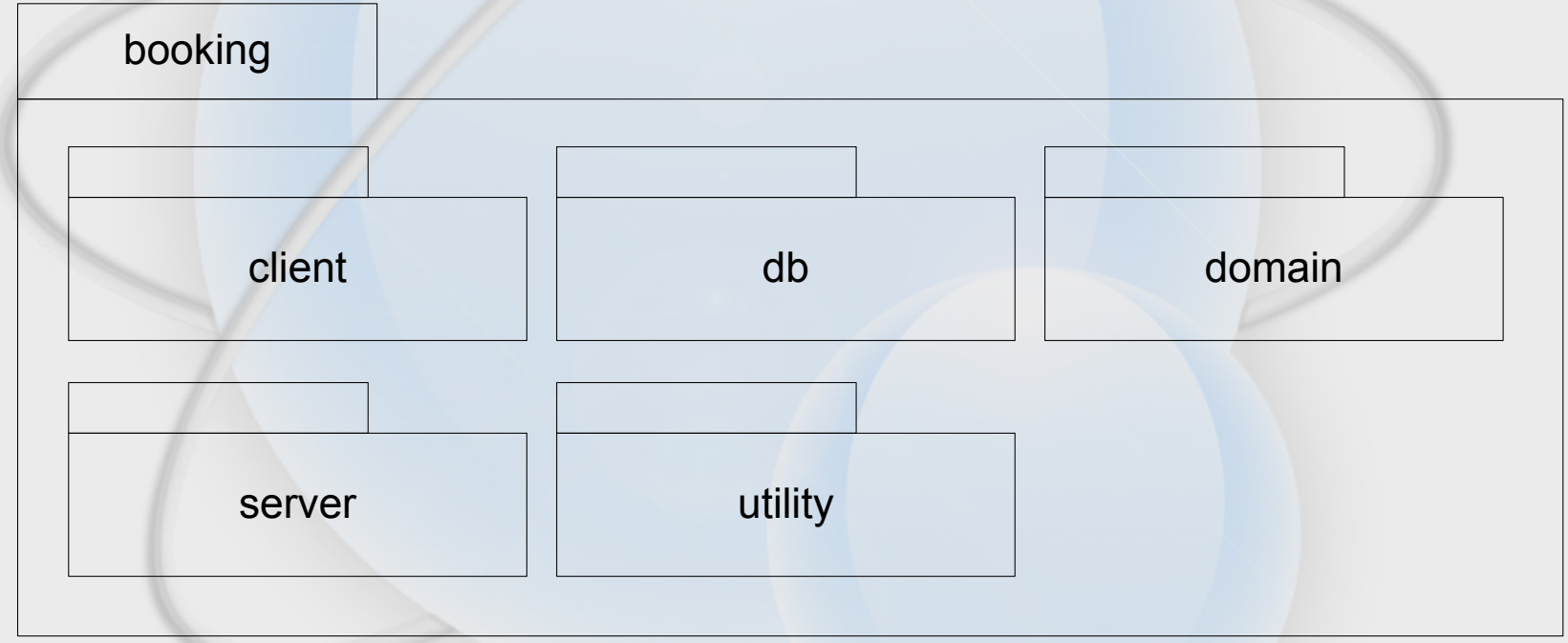
套件圖型



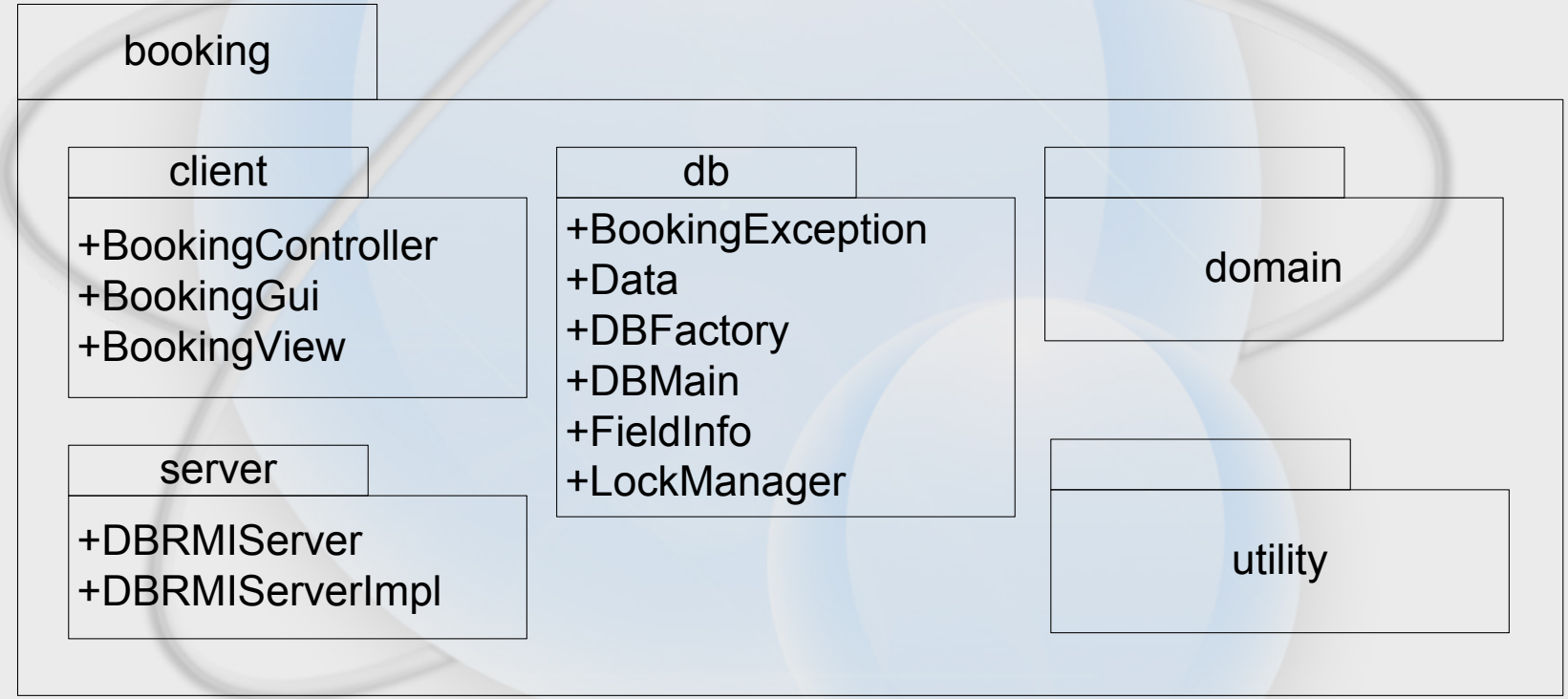
套件標記


client
<<interface>>
BookingView
BookingController
BookingGui

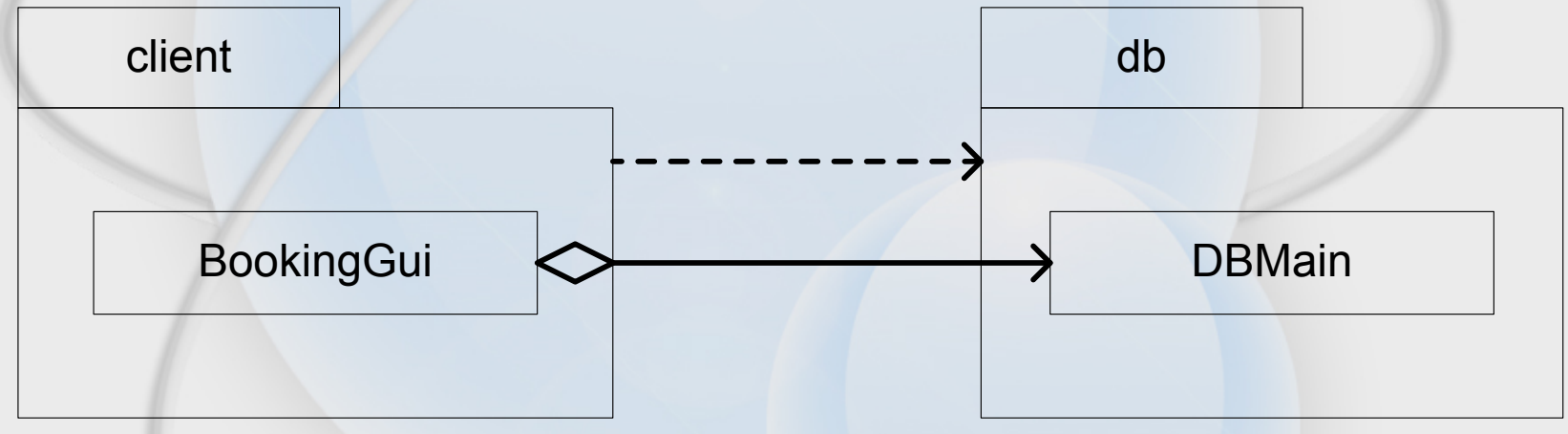
套件標記



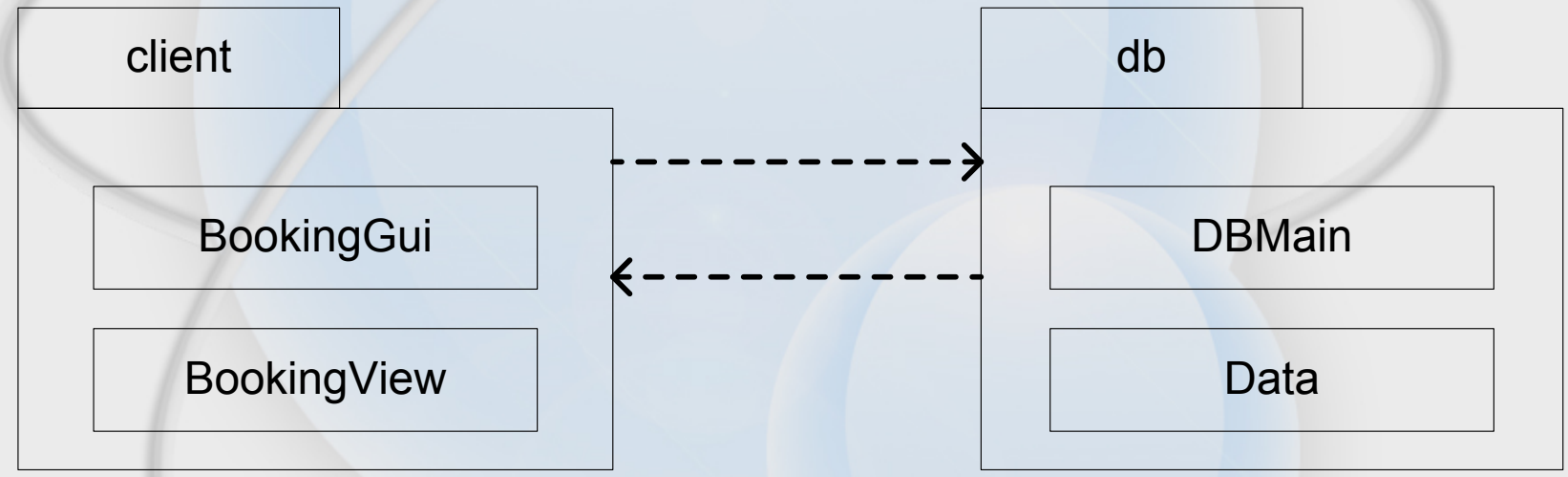
套件標記


booking
client
+BookingController
+BookingGui
+BookingView
server
+DBRMIServer
+DBRMIServerImpl
db
+BookingException
+Data
+DBFactory
+DBMain
+FieldInfo
+LockManager
domain
utility

相依性



循環相依



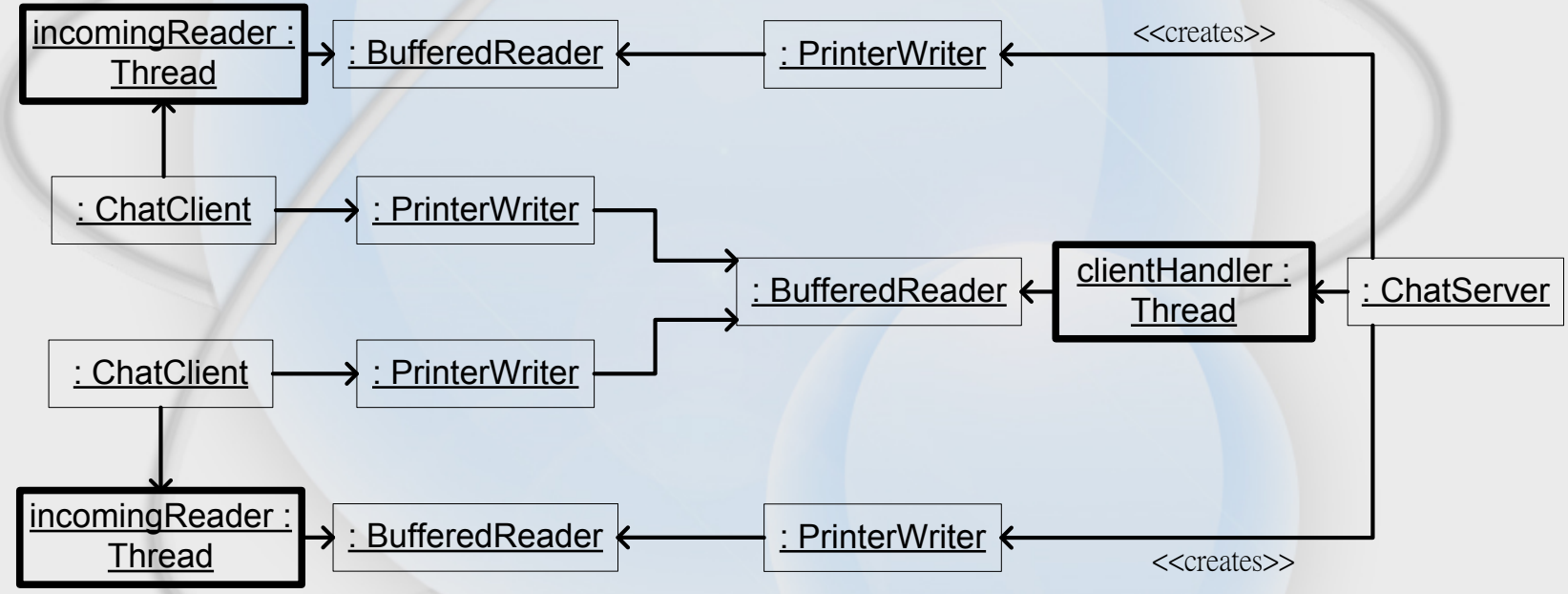
第四章

物件圖型

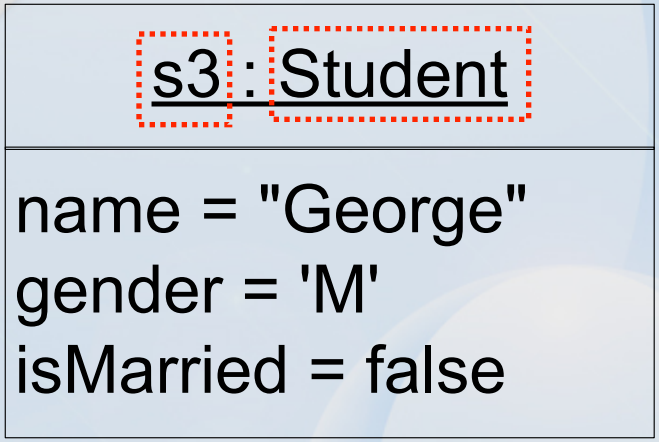
本章重點

- 物件節點
- 組合關係

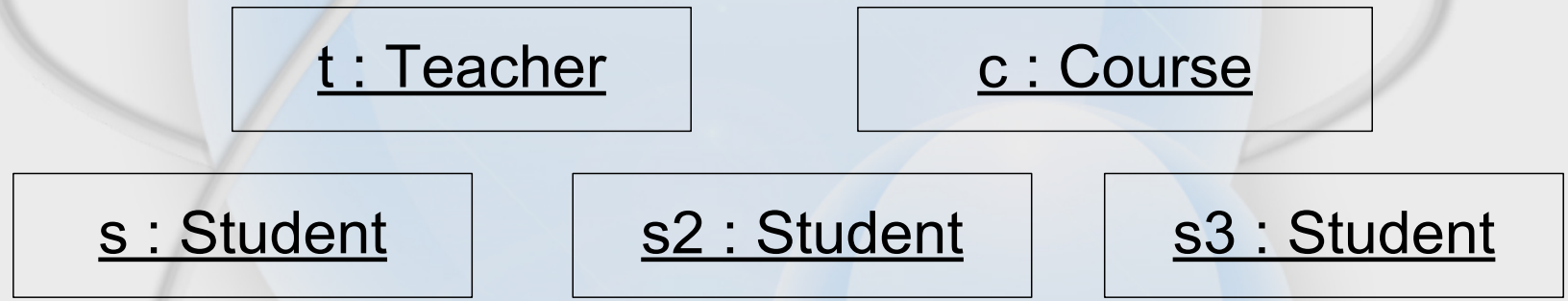
物件圖型


incomingReader : Thread
: BufferedReader
: PrinterWriter
<<creates>>
: ChatClient
: PrinterWriter
: BufferedReader
clientHandler : Thread
: ChatServer
: ChatClient
: PrinterWriter
incomingReader : Thread
: BufferedReader
: PrinterWriter
<<creates>>

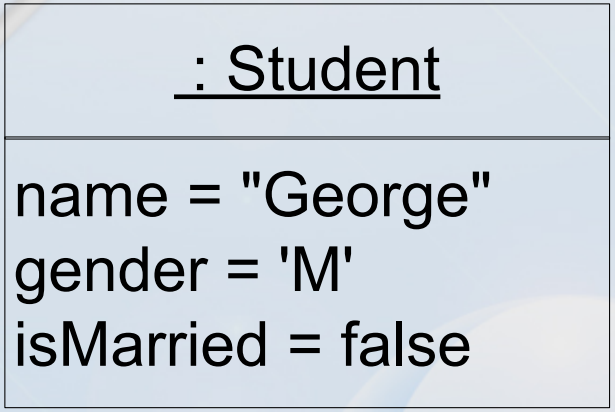
物件節點



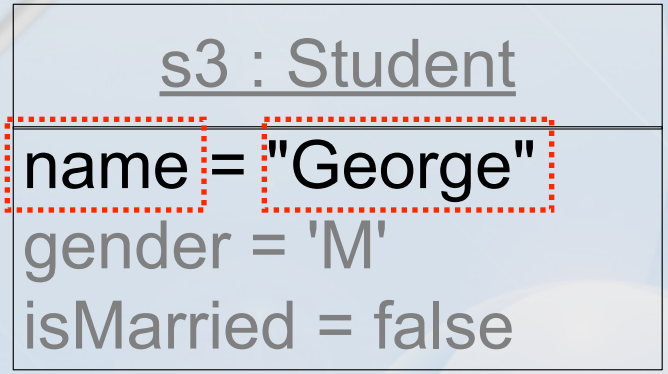
物件節點



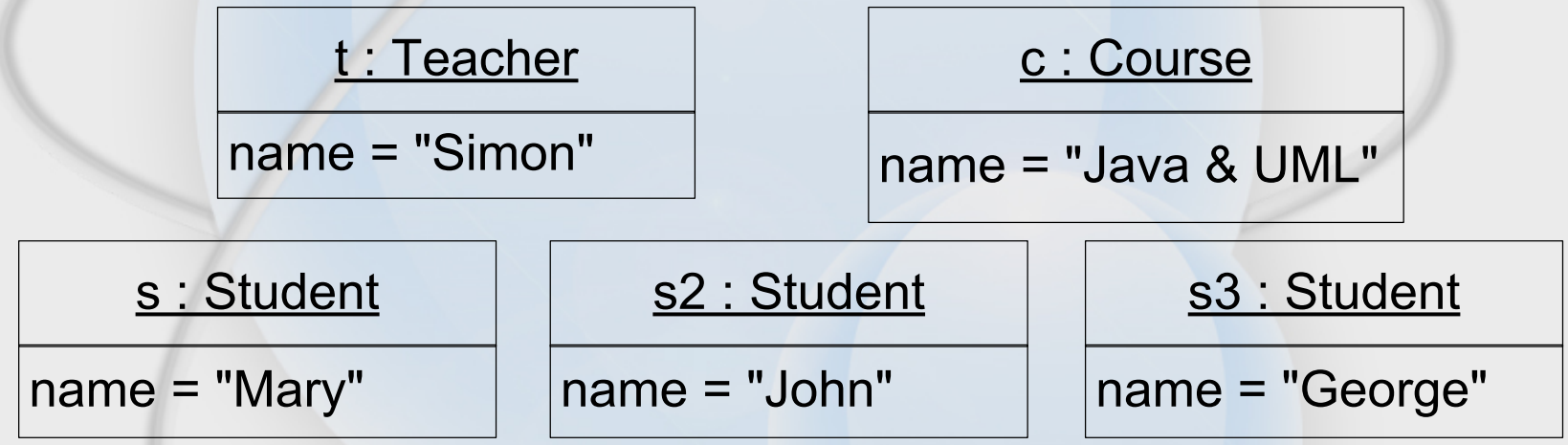
匿名物件



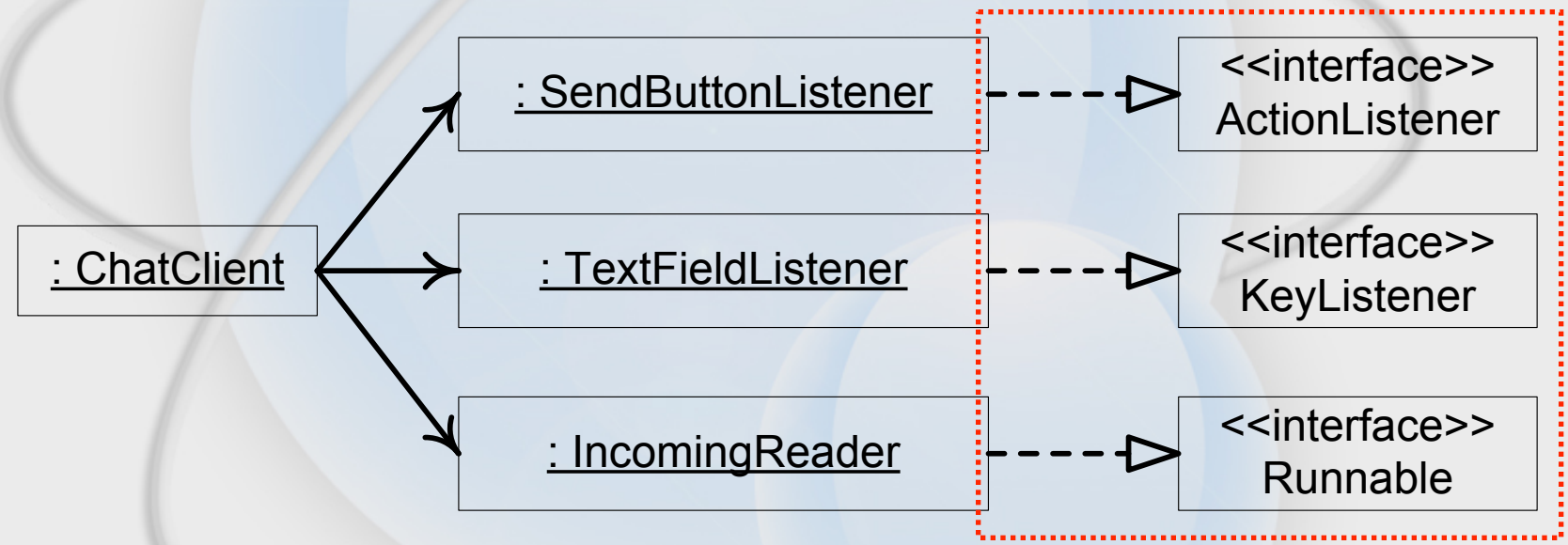
屬性區格



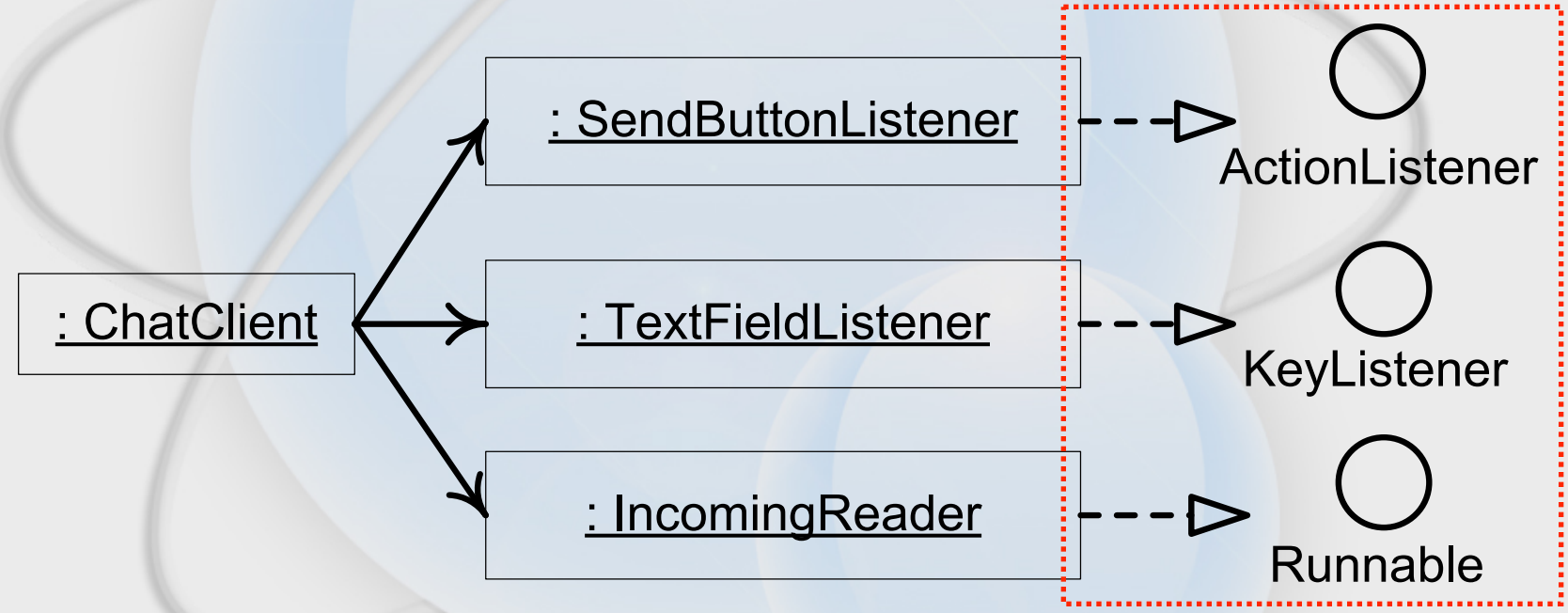
屬性區格


t : Teacher
name = "Simon"
c : Course
name = "Java & UML"
s : Student
name = "Mary"
s2 : Student
name = "John"
s3 : Student
name = "George"

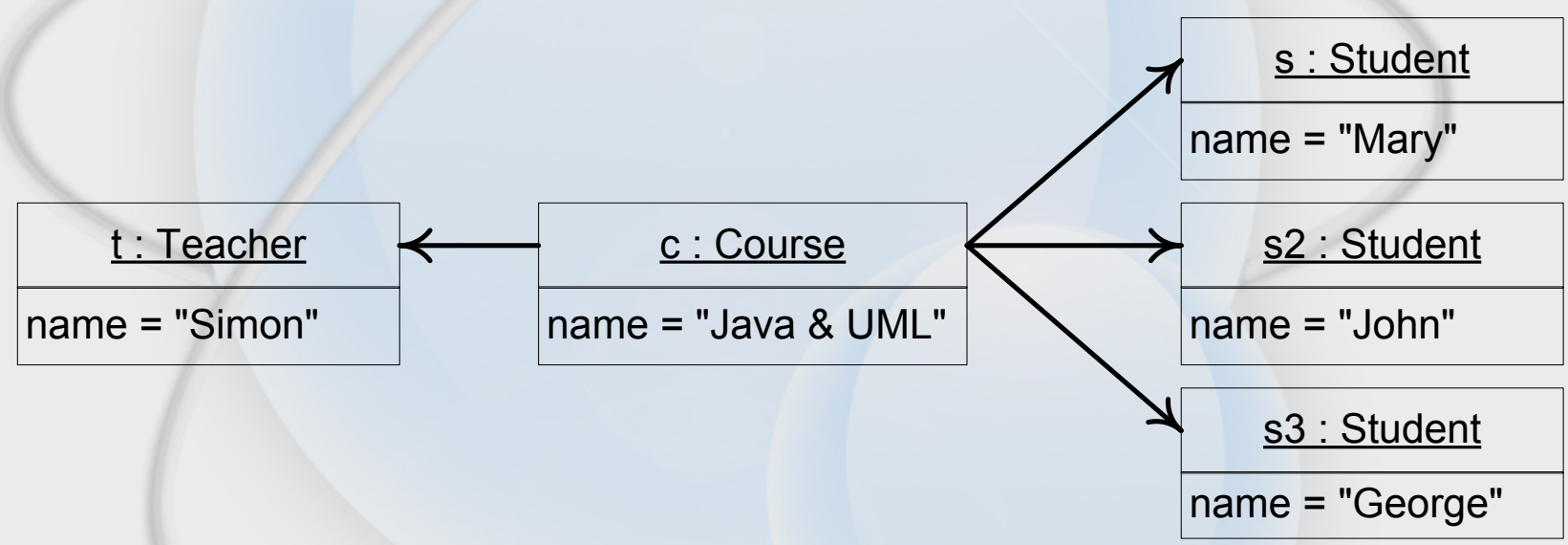
實作


: ChatClient
: SendButtonListener
: TextFieldListener
: IncomingReader
<<interface>>
ActionListener
<<interface>>
KeyListener
<<interface>>
Runnable

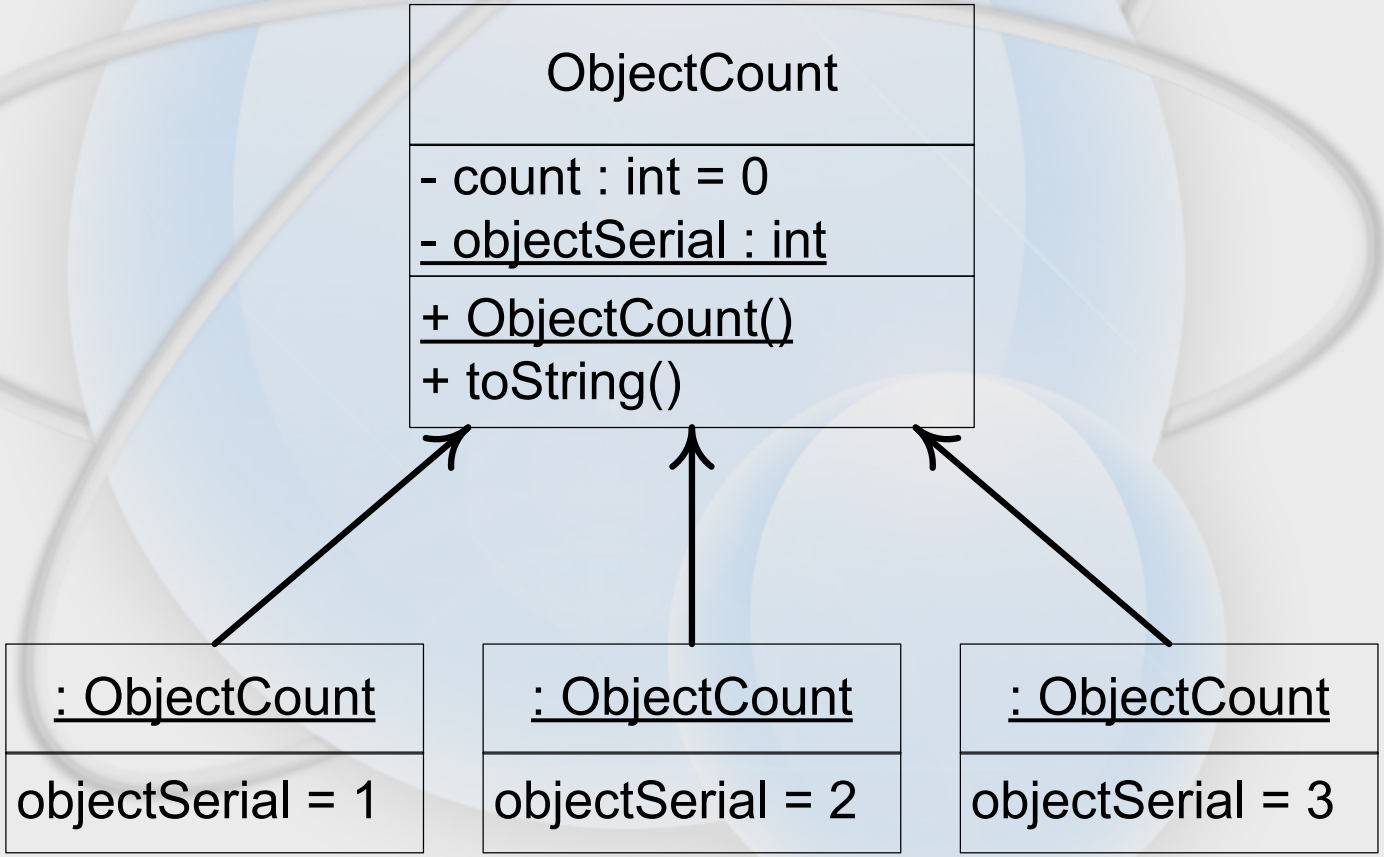
實作


: ChatClient
: SendButtonListener
: TextFieldListener
: IncomingReader
ActionListener
KeyListener
Runnable

結合


t : Teacher
name = "Simon"
c : Course
name = "Java & UML"
s : Student
name = "Mary"
s2 : Student
name = "John"
s3 : Student
name = "George"

類別屬性


ObjectCount
- count : int = 0
- objectSerial : int
+ ObjectCount()
+ toString()
: ObjectCount
objectSerial = 1
: ObjectCount
objectSerial = 2
: ObjectCount
objectSerial = 3

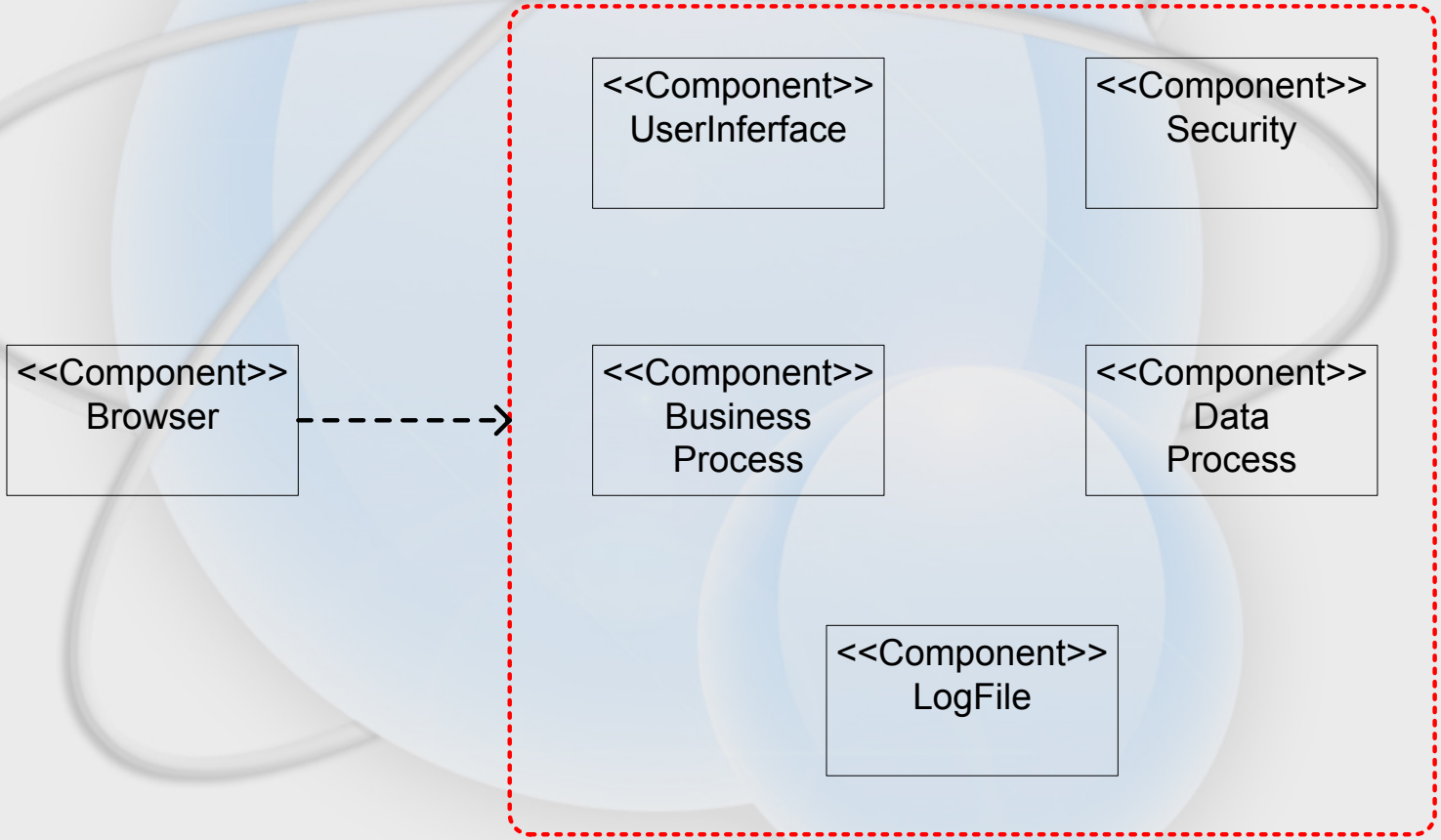
第五章

元件圖型

本章重點

- 元件
- 元件與介面
- 使用元件圖型

元件圖型


<<Component>>
UserInferface
<<Component>>
Security
<<Component>>
Browser
<<Component>>
Business
Process
<<Component>>
Data
Process
<<Component>>
LogFile

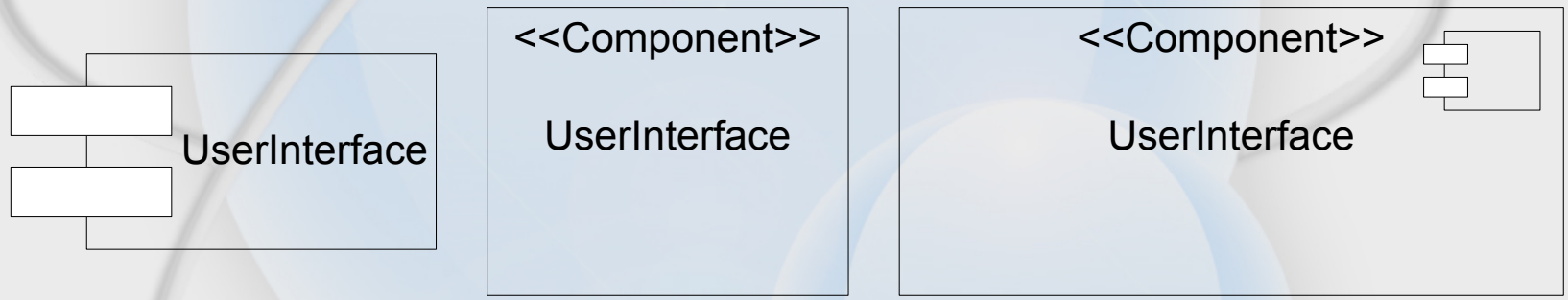
元件

- 使用者可以預先瞭解系統功能的架構
- 提供軟體系統功能的邏輯性文件
- 提供更良好的封裝
- 方便取代與重複使用

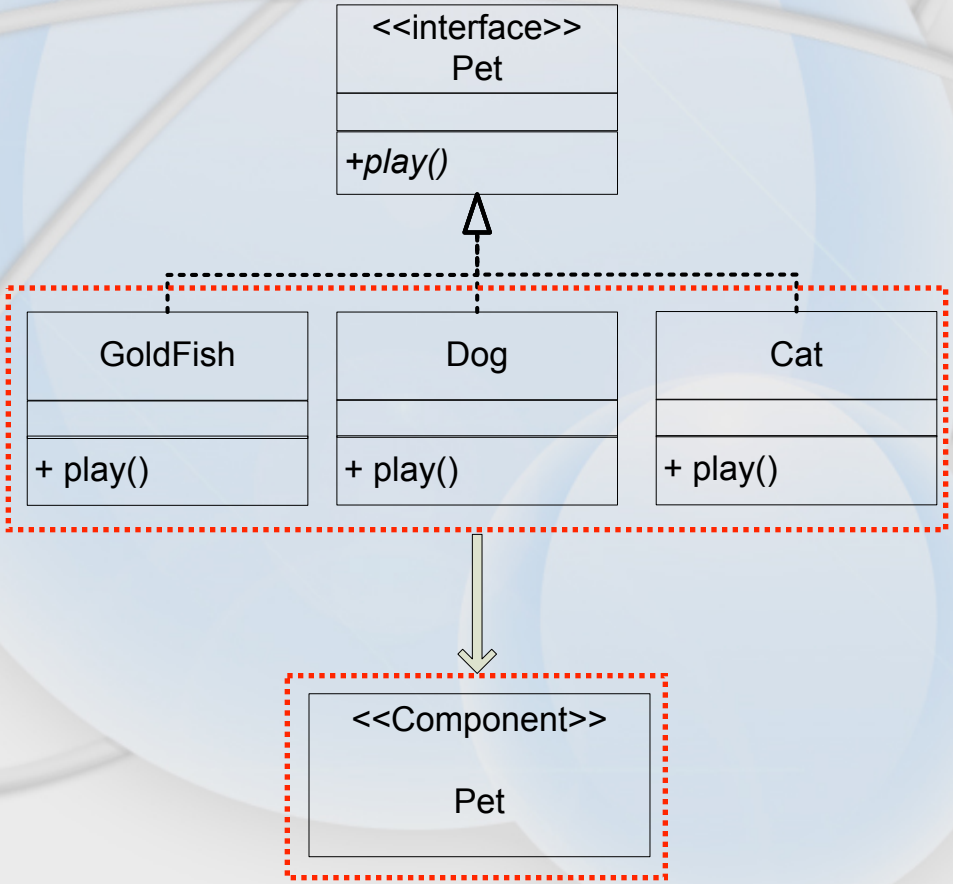
元件的分類

- 佈署元件
- 工作產物元件
- 可執行元件

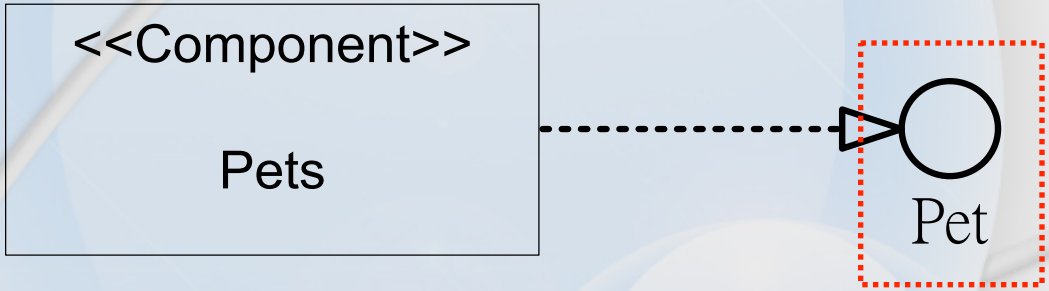
元件表示方法


UserInterface
<<Component>>
UserInterface
<<Component>>
UserInterface

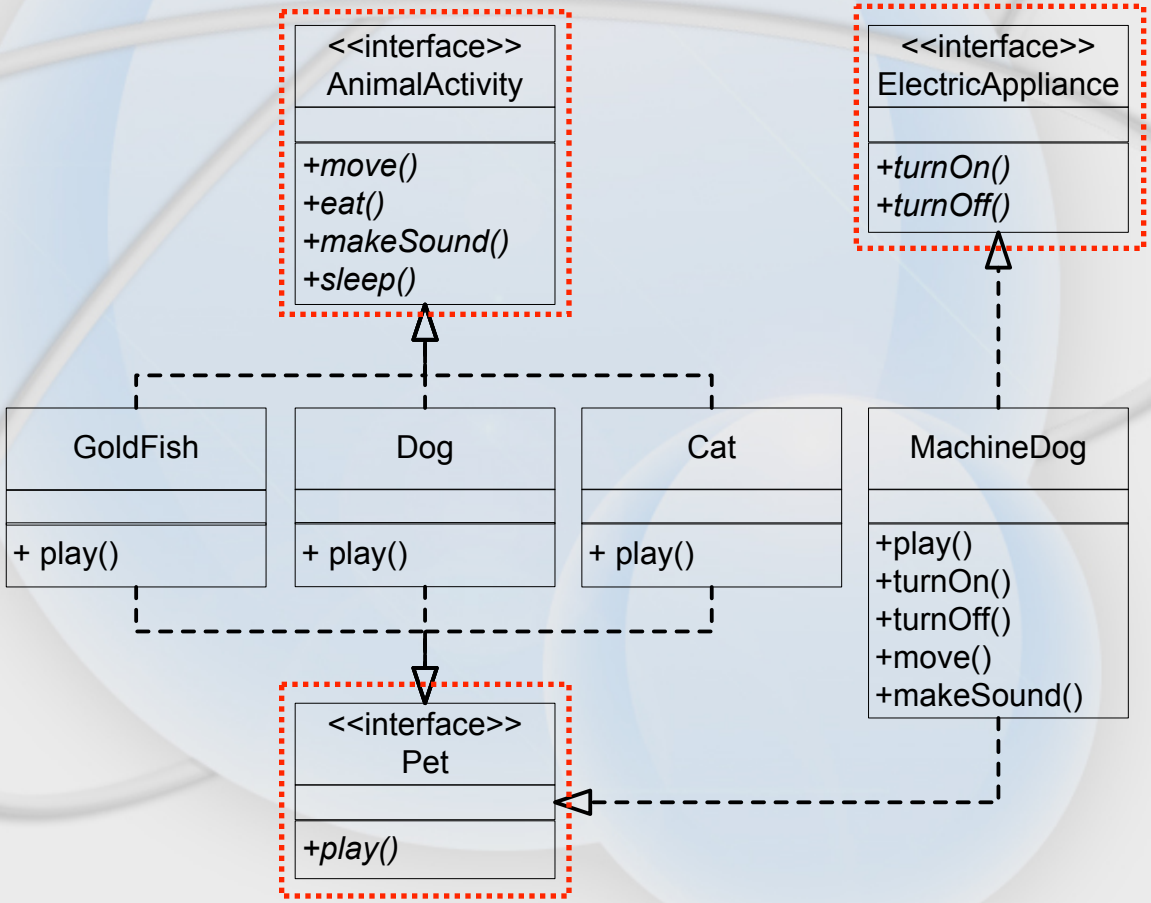
元件與介面


<<interface>>
Pet
+play()
GoldFish
+ play()
Dog
+ play()
Cat
+ play()
<<Component>>
Pet

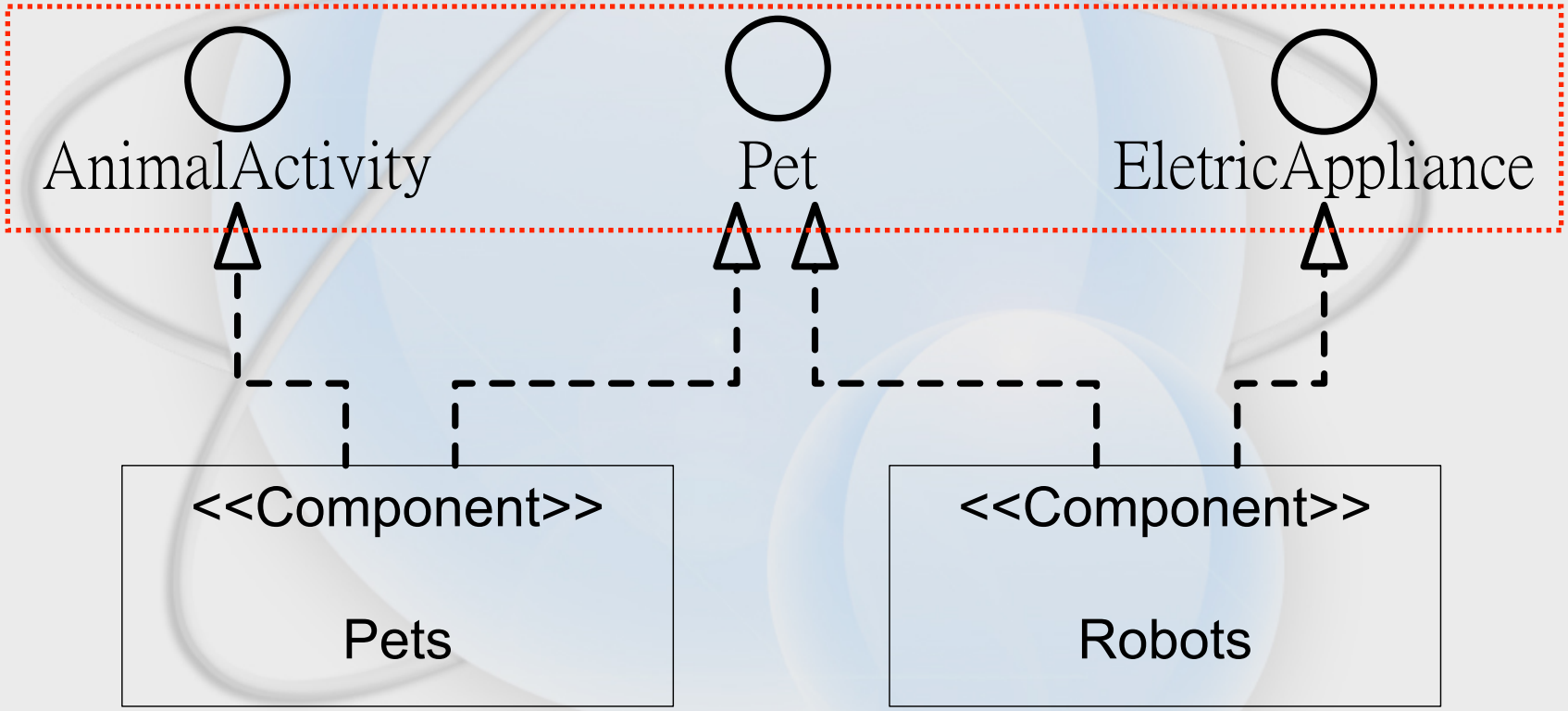
元件與介面



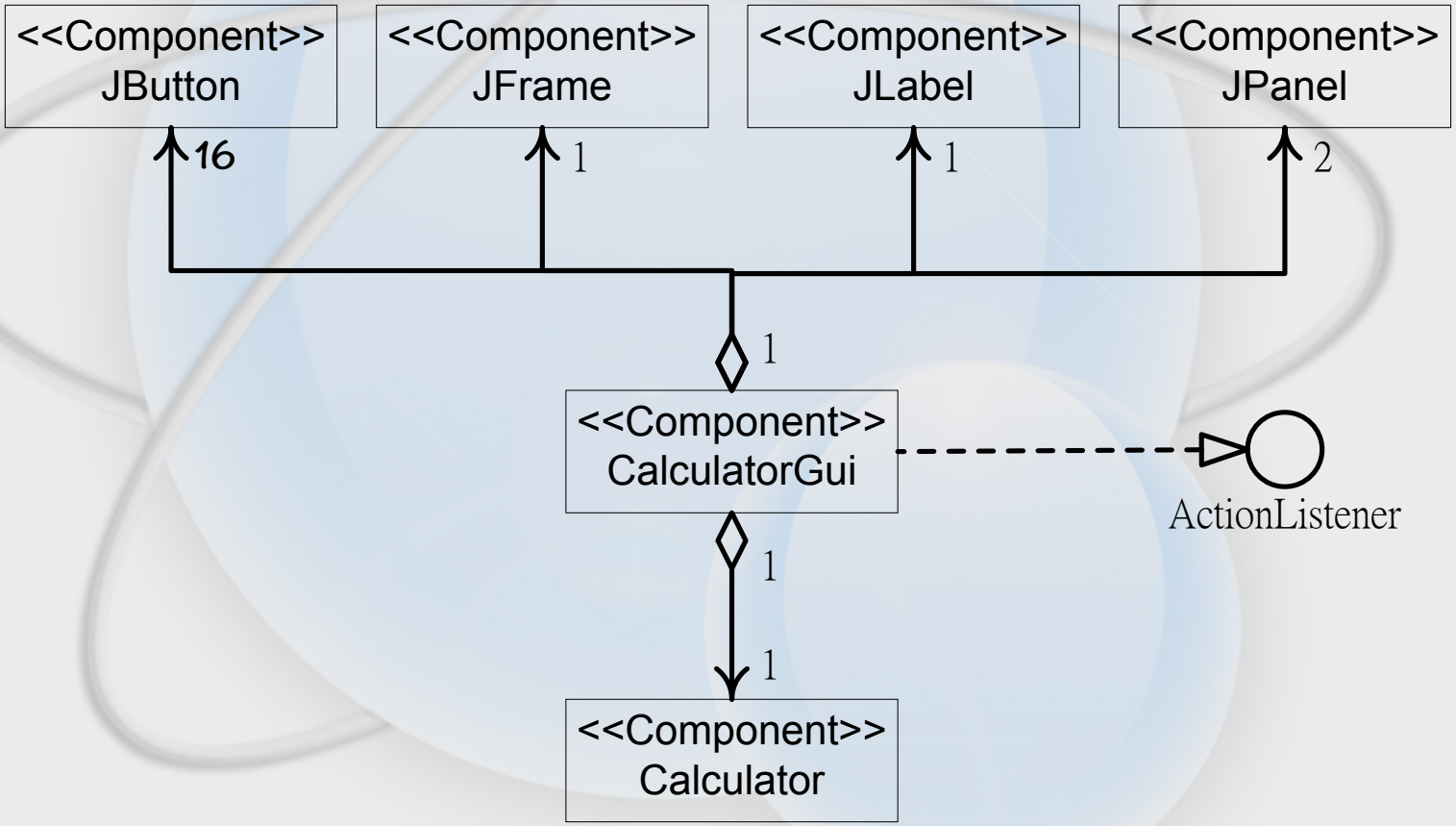
元件與介面


<<interface>>
AnimalActivity
+move()
+eat()
+makeSound()
+sleep()
<<interface>>
ElectricAppliance
+turnOn()
+turnOff()
GoldFish
+ play()
Dog
+ play()
Cat
+ play()
MachineDog
+play()
+turnOn()
+turnOff()
+move()
+makeSound()
<<interface>>
Pet
+play()

元件與介面


AnimalActivity
Pet
EletricAppliance
<<Component>>
Pets
<<Component>>
Robots

元件架構


<<Component>>
JButton
<<Component>>
JFrame
<<Component>>
JLabel
<<Component>>
JPanel
16
1
1
2
1
<<Component>>
CalculatorGui
ActionListener
1
1
<<Component>>
Calculator

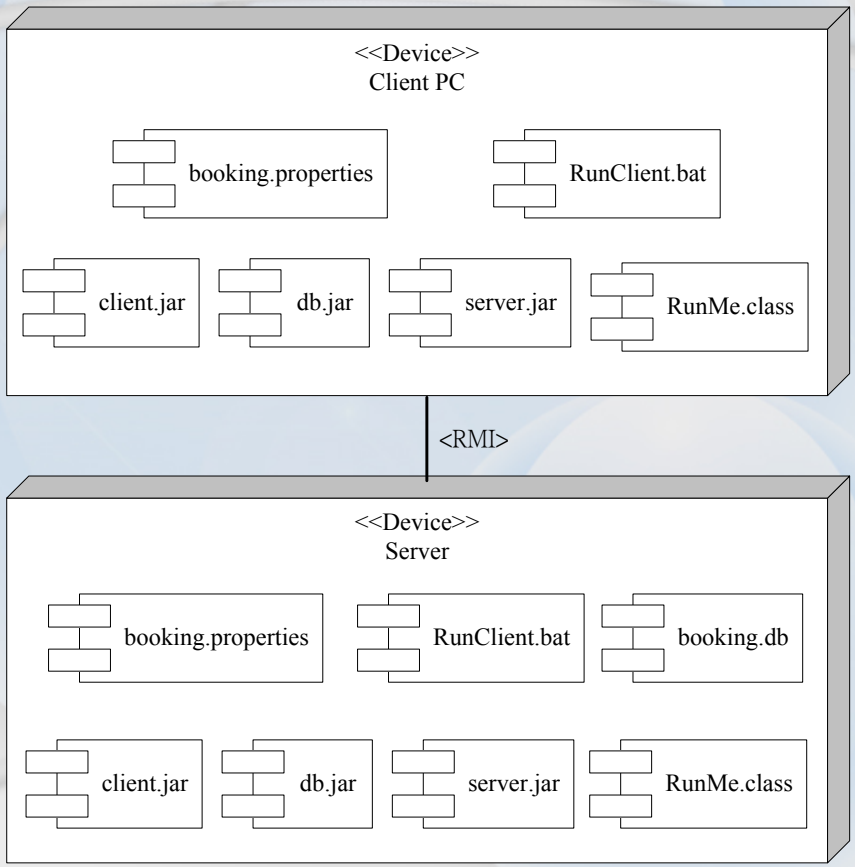
第六章

佈署圖型

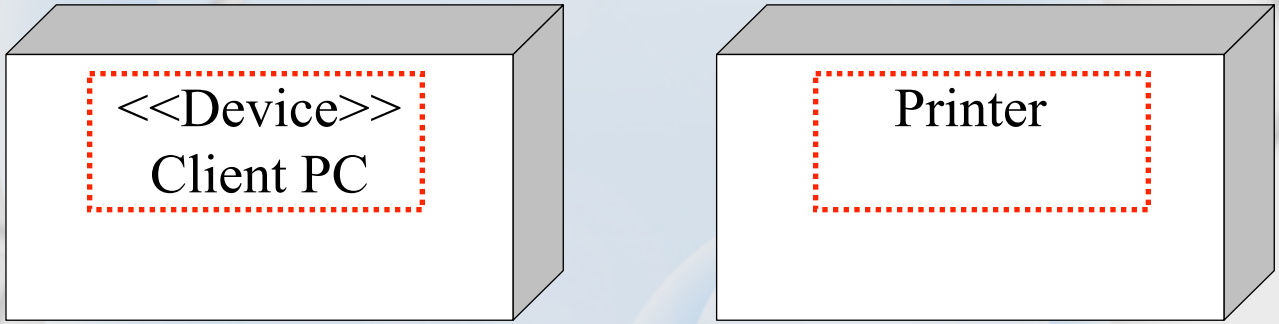
本章重點

- 節點
- 關連

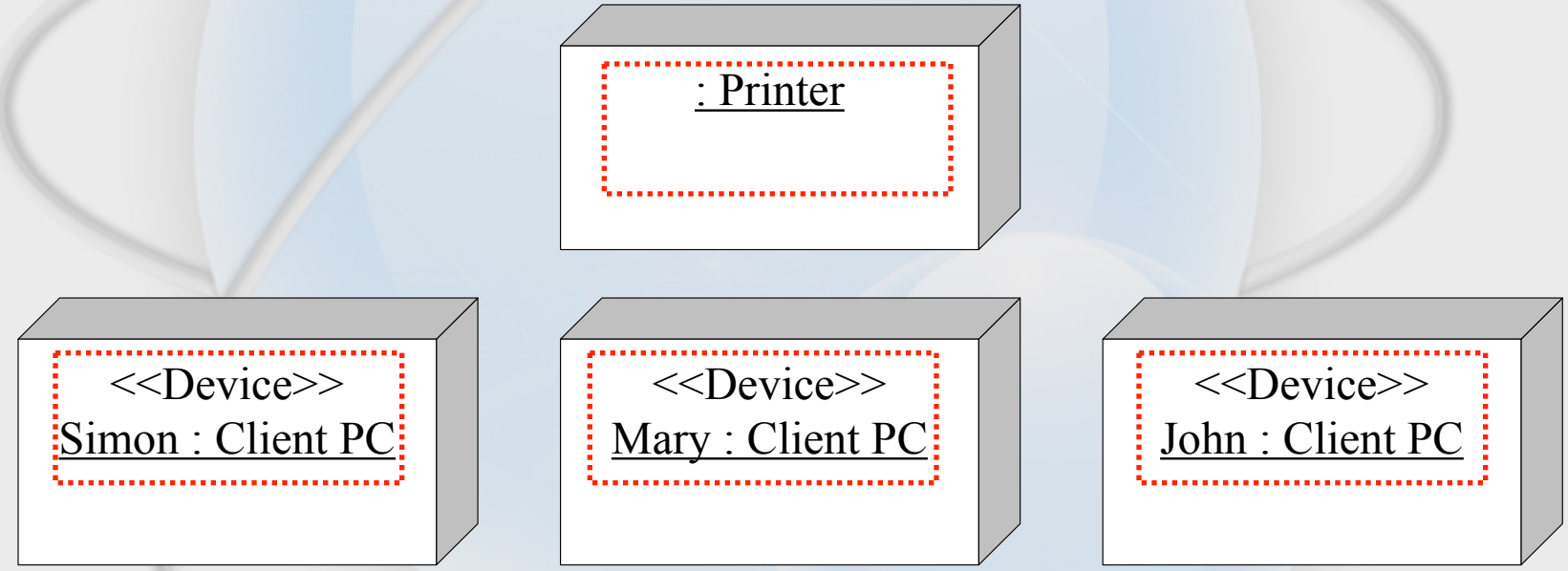
佈署圖型


<<Device>>
Client PC
booking.properties
RunClient.bat
client.jar
db.jar
server.jar
RunMe.class
<RMI>
<<Device>>
Server
booking.properties
RunClient.bat
booking.db
client.jar
db.jar
server.jar
RunMe.class

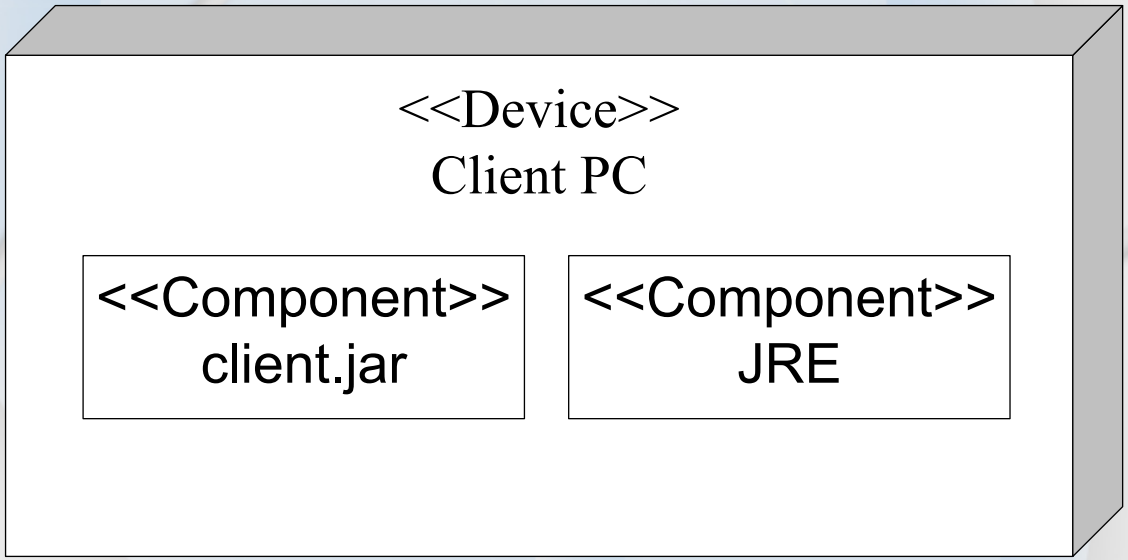
硬體節點



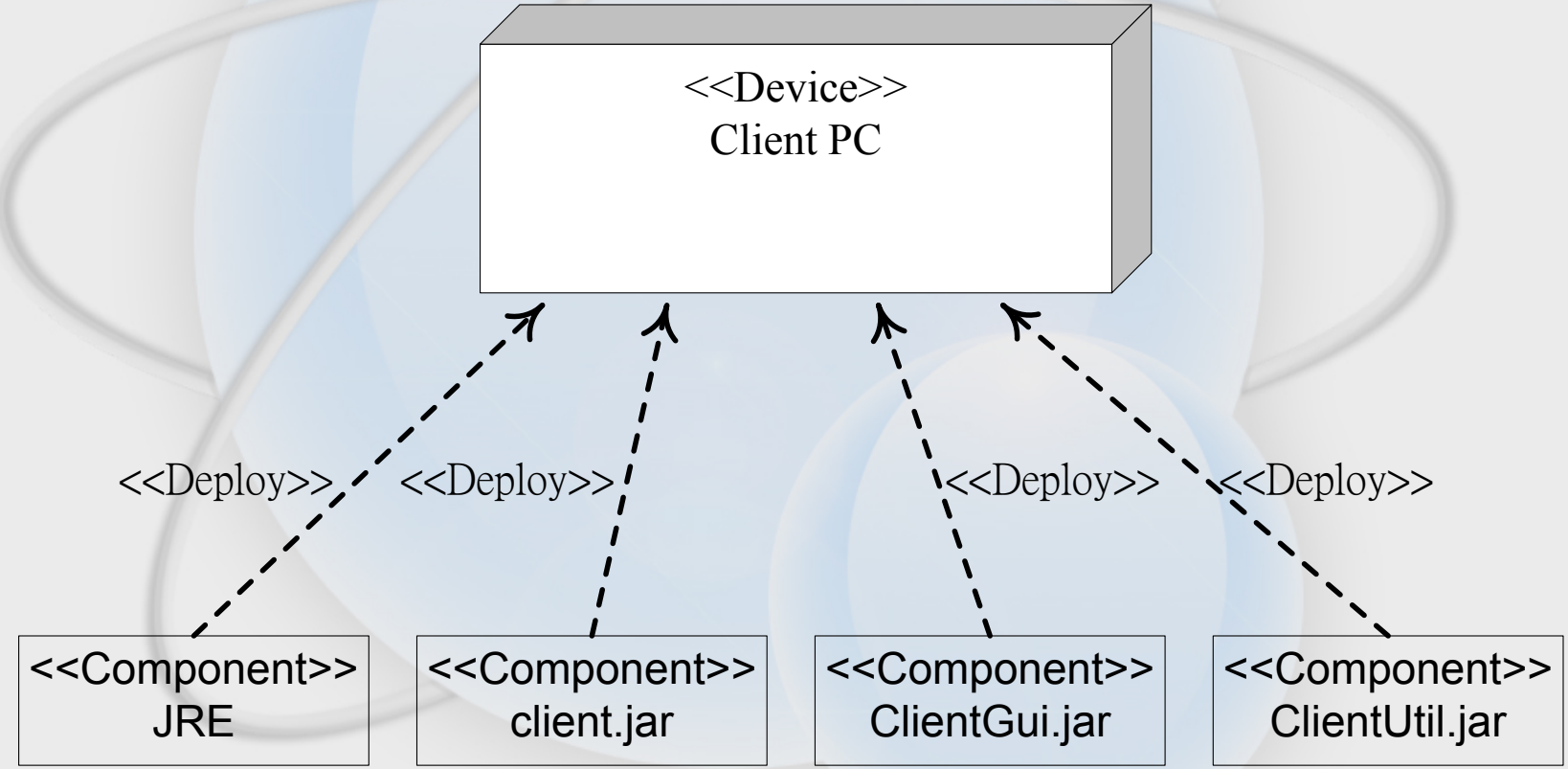
實體節點


: Printer
<<Device>>
Simon : Client PC
<<Device>>
Mary : Client PC
<<Device>>
John : Client PC

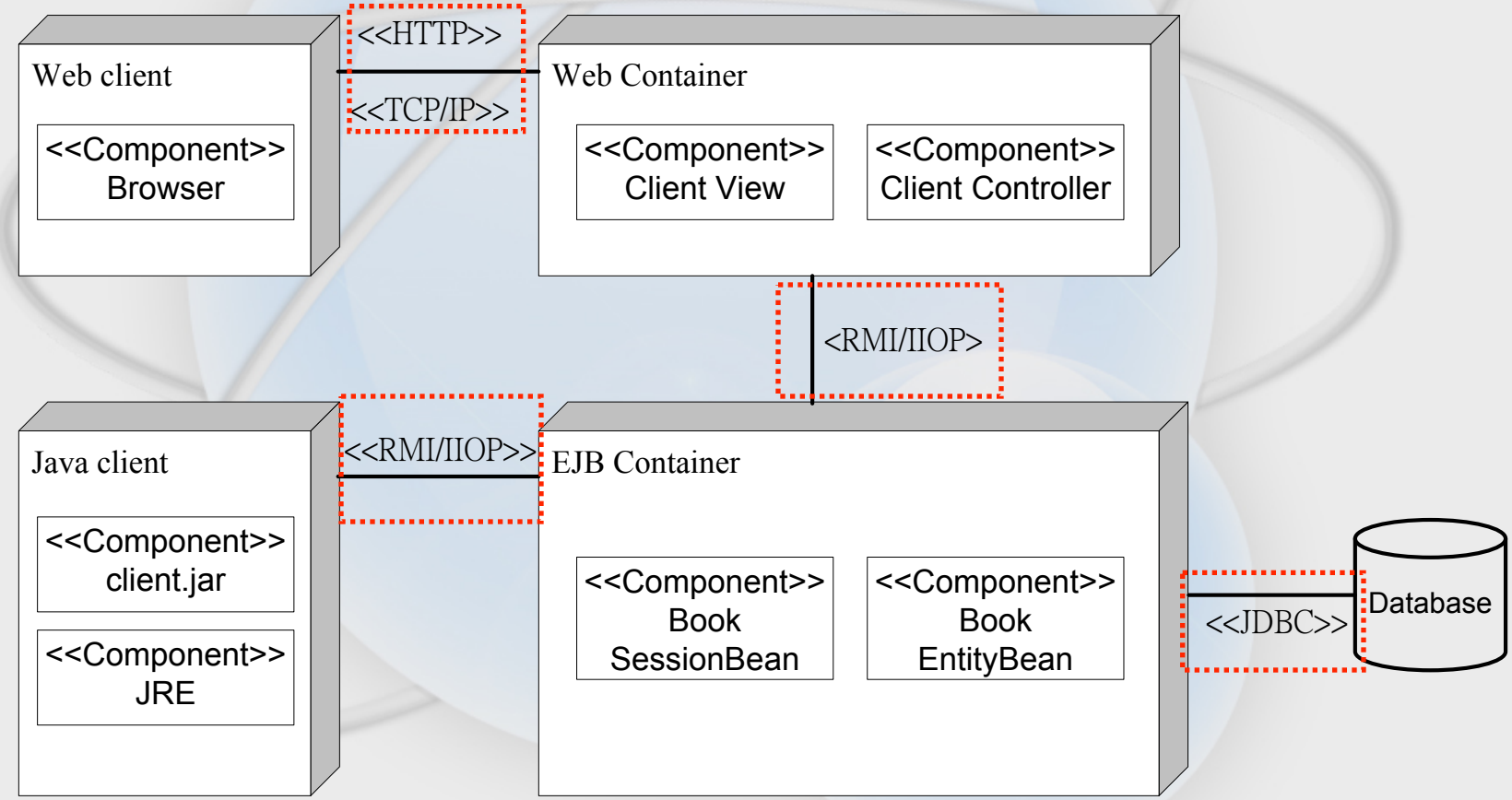
使用元件節點



使用元件節點


<<Device>>
Client PC
<<Deploy>>
<<Deploy>>
<<Deploy>>
<<Deploy>>
<<Component>>
JRE
<<Component>>
client.jar
<<Component>>
ClientGui.jar
<<Component>>
ClientUtil.jar

關連


Web client
<<Component>>
Browser
<<HTTP>>
<<TCP/IP>>
Web Container
<<Component>>
Client View
<<Component>>
Client Controller
<RMI/IIOP>
Java client
<<Component>>
client.jar
<<Component>>
JRE
<<RMI/IIOP>>
EJB Container
<<Component>>
Book
SessionBean
<<Component>>
Book
EntityBean
<<JDBC>>
Database

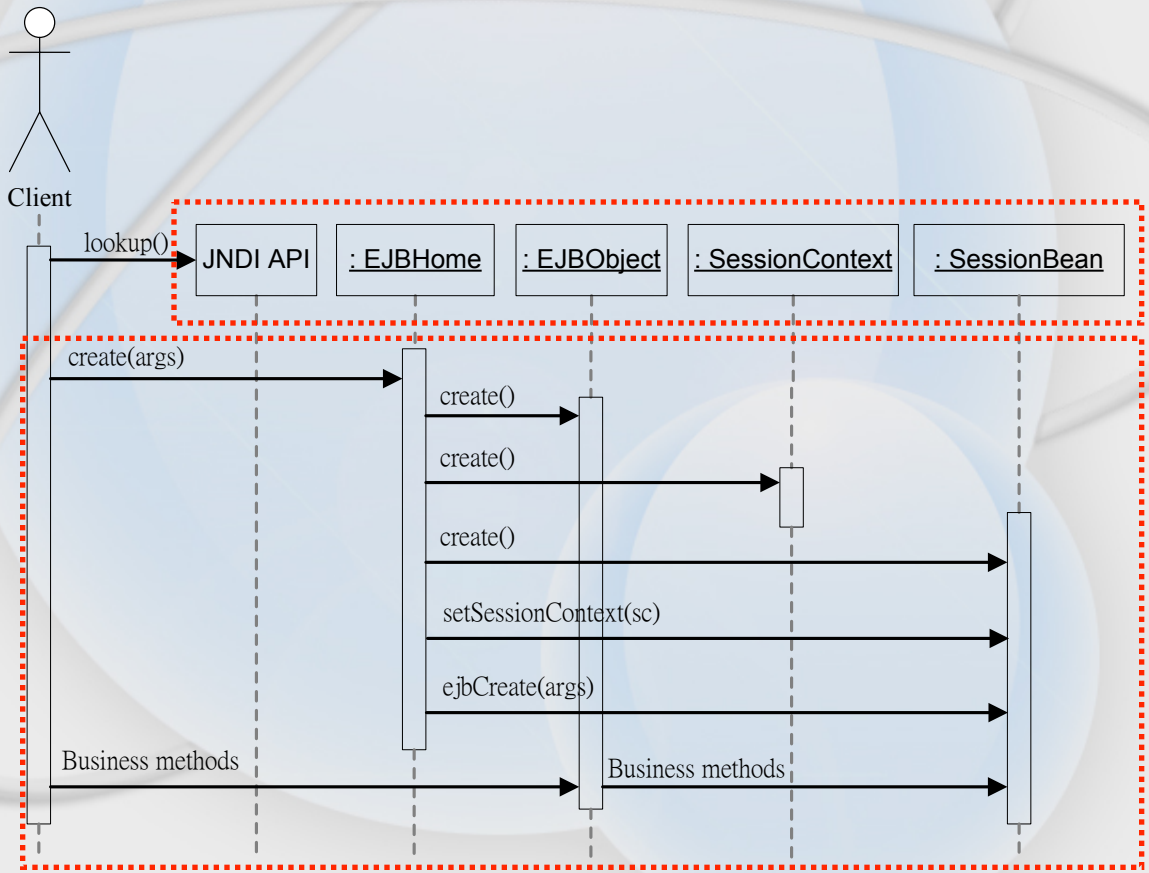
第七章

循序圖型

本章重點

- 元素
- 多執行緒

循序圖型


Client
lookup()
JNDI API
: EJBHome
: EJBObject
: SessionContext
: SessionBean
create(args)
create()
create()
create()
setSessionContext(sc)
ejbCreate(args)
Business methods
Business methods

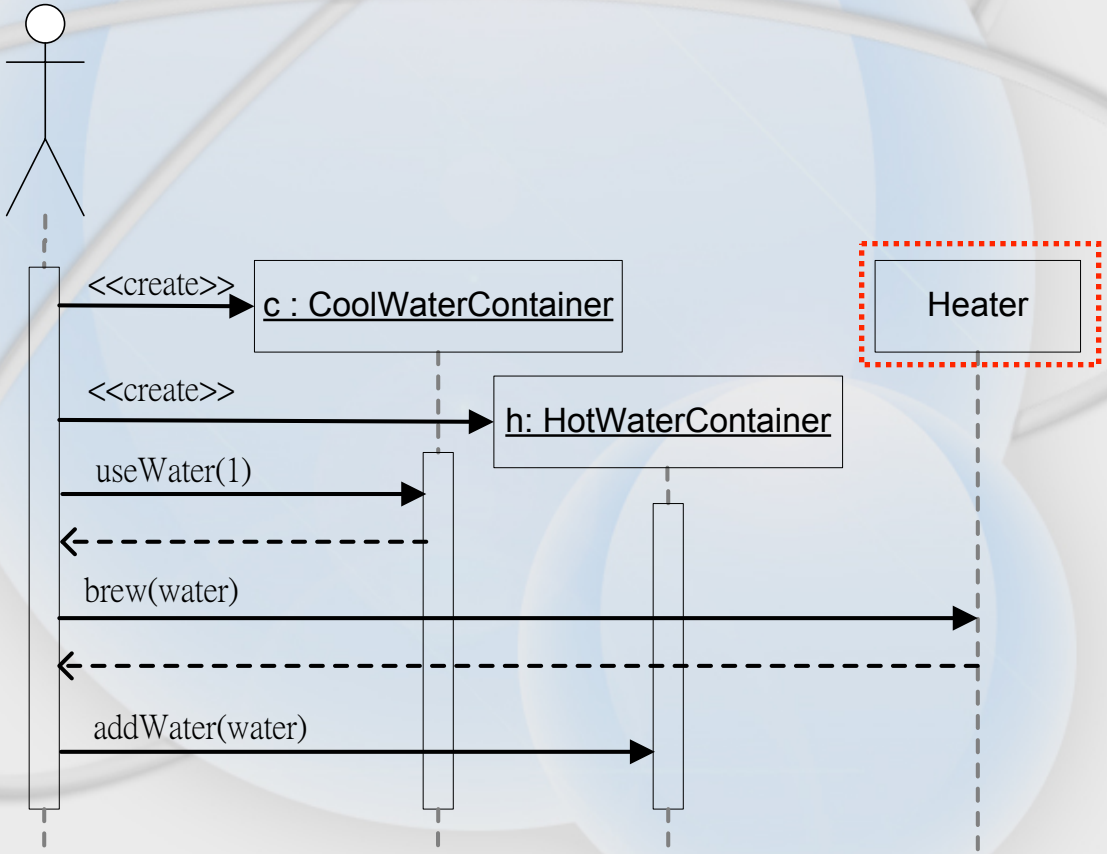
元素

- 物件節點(Object node)
- 生命線(Lifeline)
- 活化區塊(Activation box)
- 訊息(Message)
- 內部訊息
- 解構物件
- 迴圈

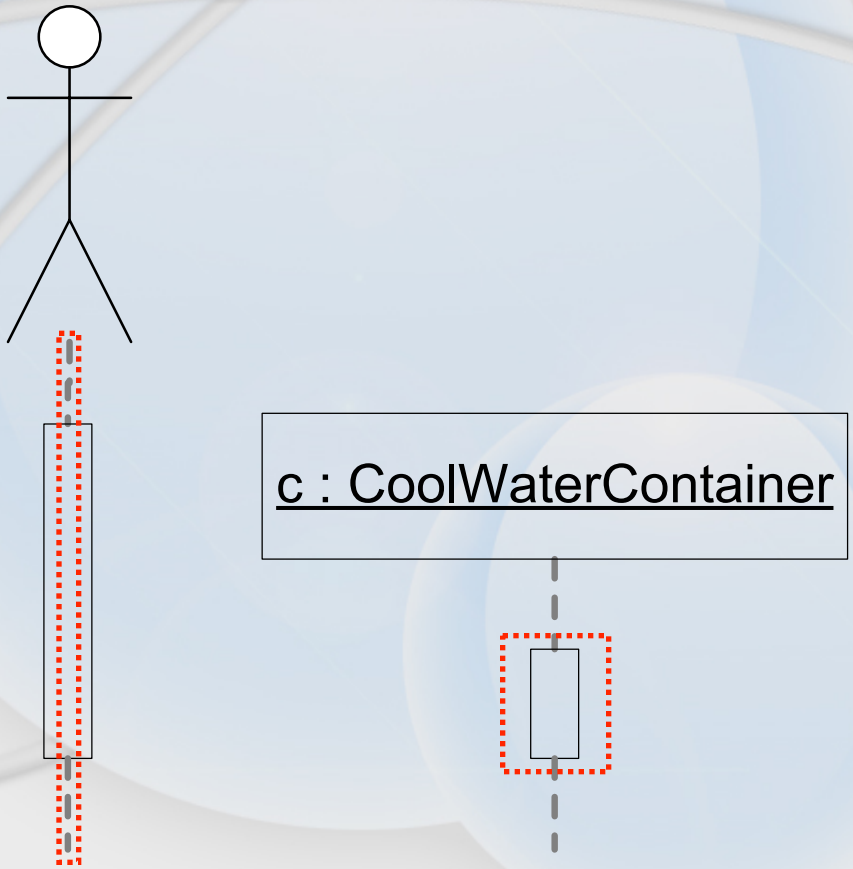
物件節點


TestThermos
h: HotWaterContainer
c : CoolWaterContainer
g: ThermosGui
t : Thermos

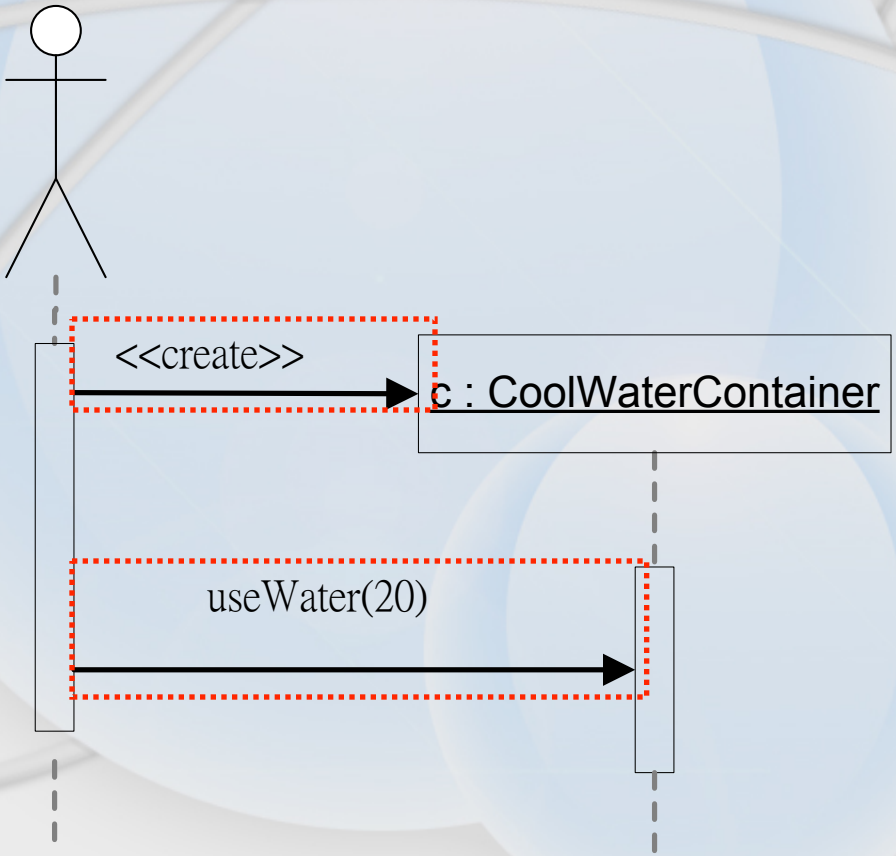
類別節點


<<create>>
c : CoolWaterContainer
Heater
<<create>>
h: HotWaterContainer
useWater(1)
brew(water)
addWater(water)

生命線與活化區塊

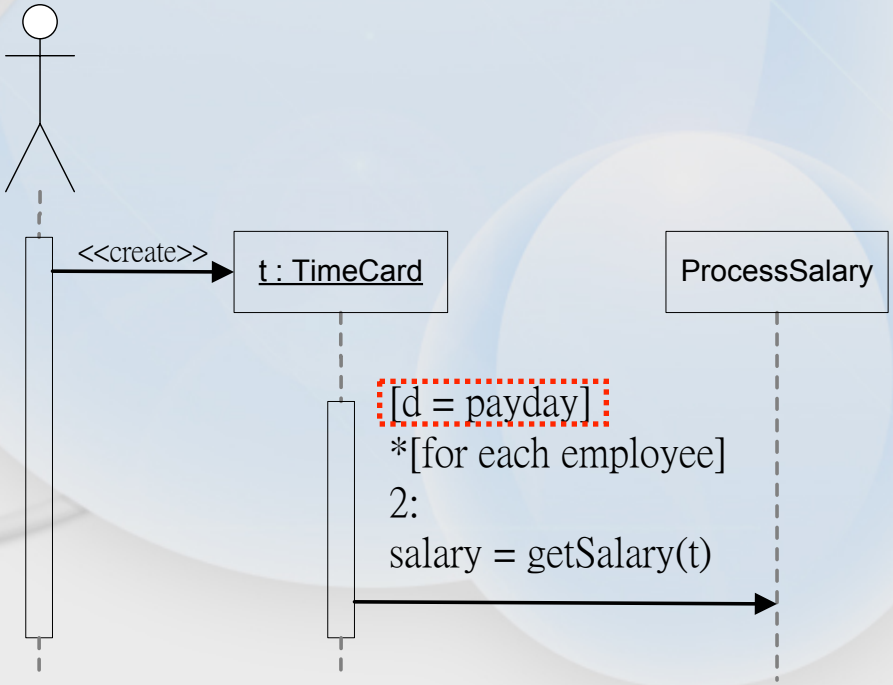


訊息



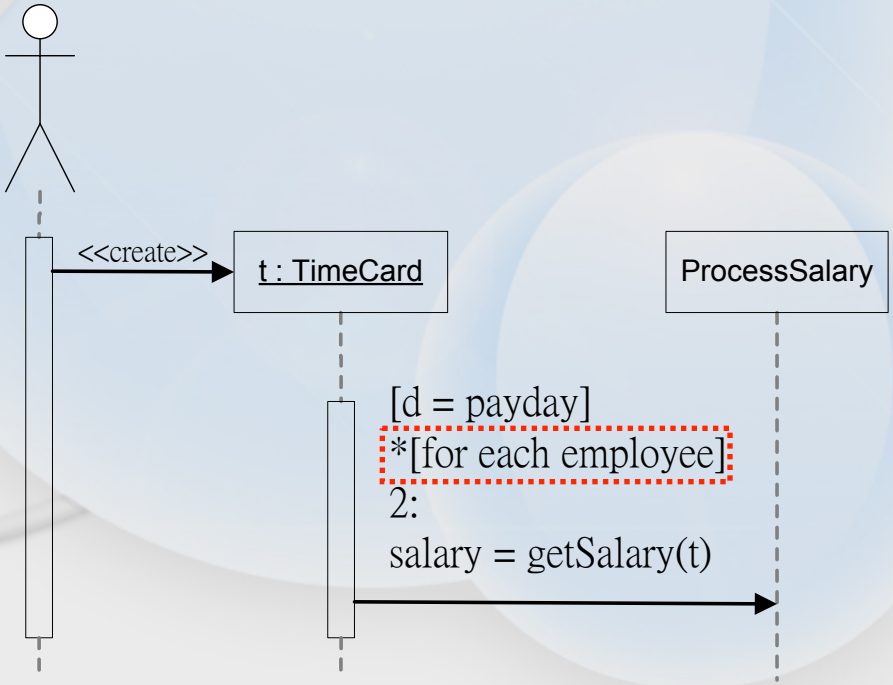
訊息宣告語法

[[<條件>]] [[*<重複>]] [<序號> :] <回傳值> := <操作名稱>(<參數>)



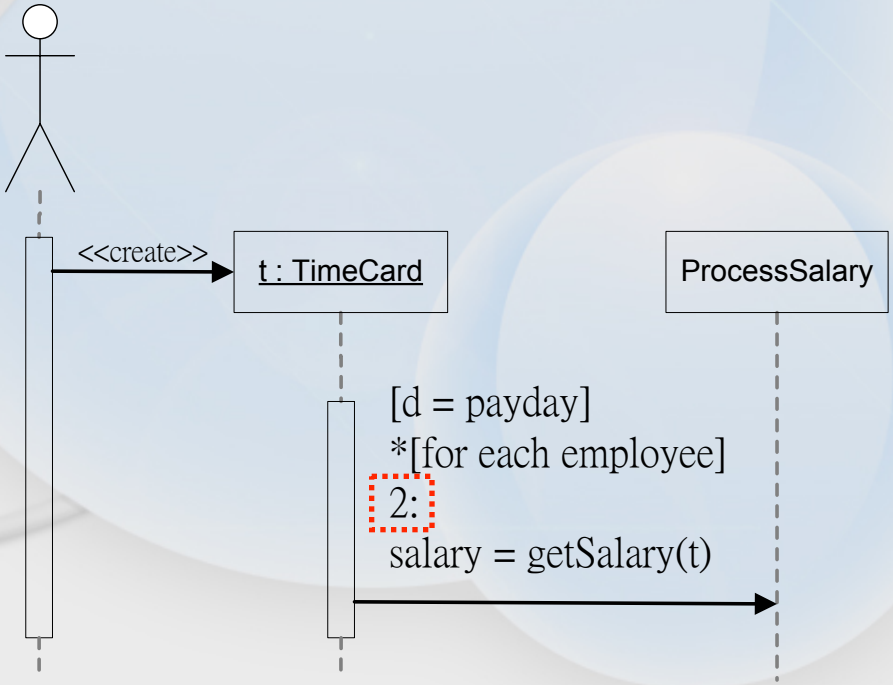
訊息宣告語法

[[<條件>]] [[*<重複>]] [<序號> :] <回傳值> := <操作名稱>(<參數>)



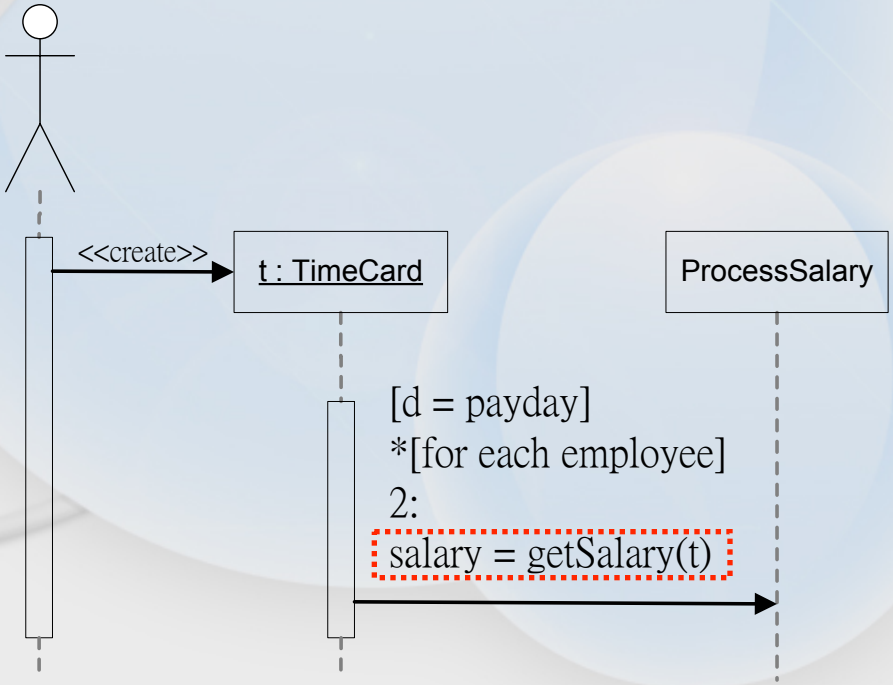
訊息宣告語法

[[<條件>]] [[*<重複>]] [<序號> :] <回傳值> := <操作名稱>(<參數>)

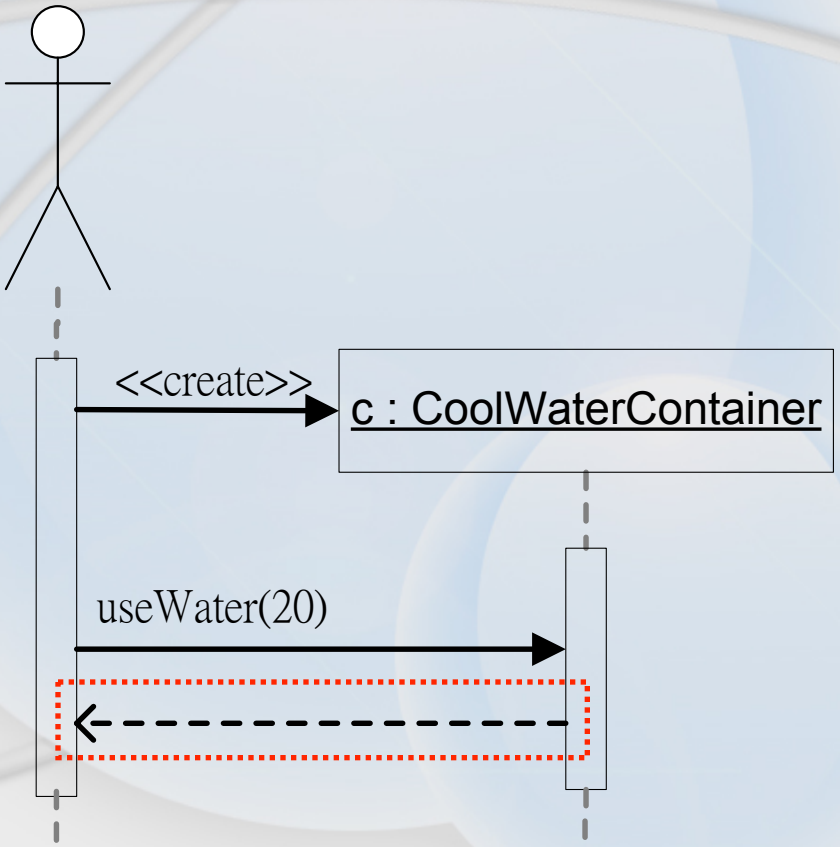


訊息宣告語法

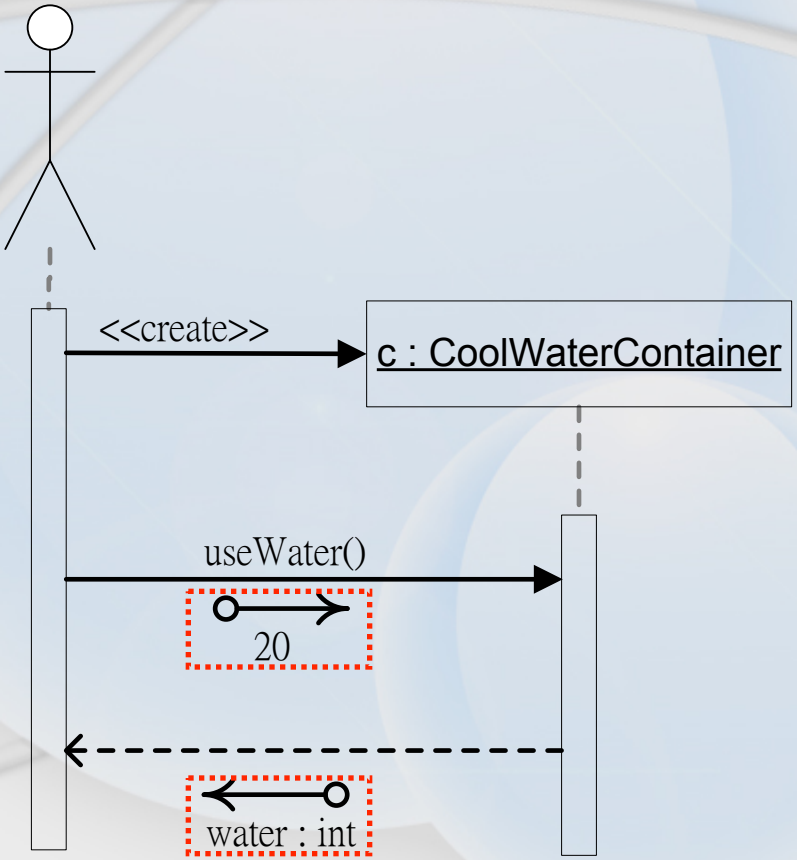
[[<條件>]] [[*<重複>]] [<序號> :] <回傳值> := <操作名稱>(<參數>)



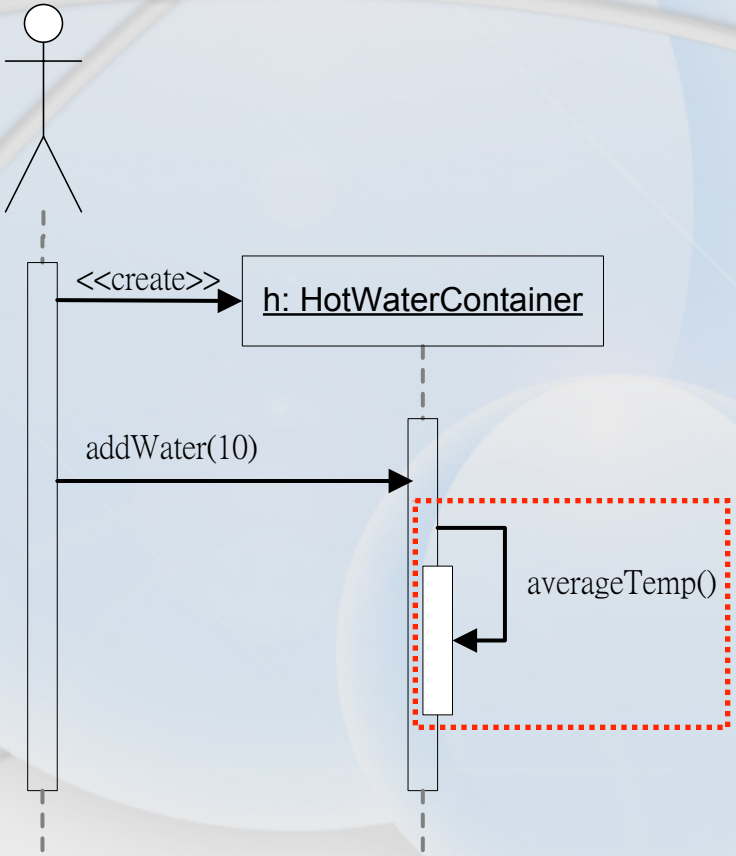
回傳值表示方法



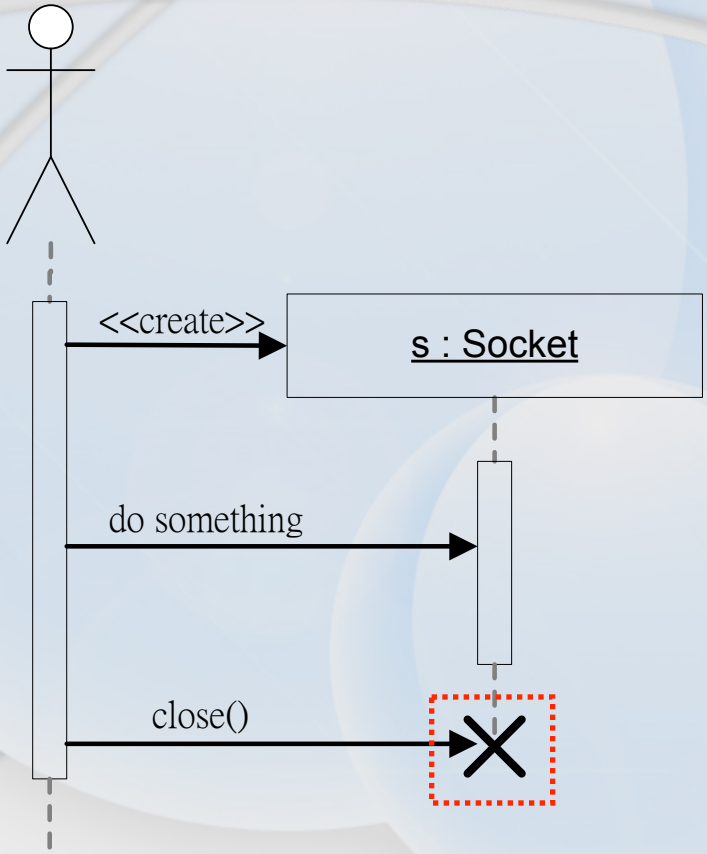
資料標記



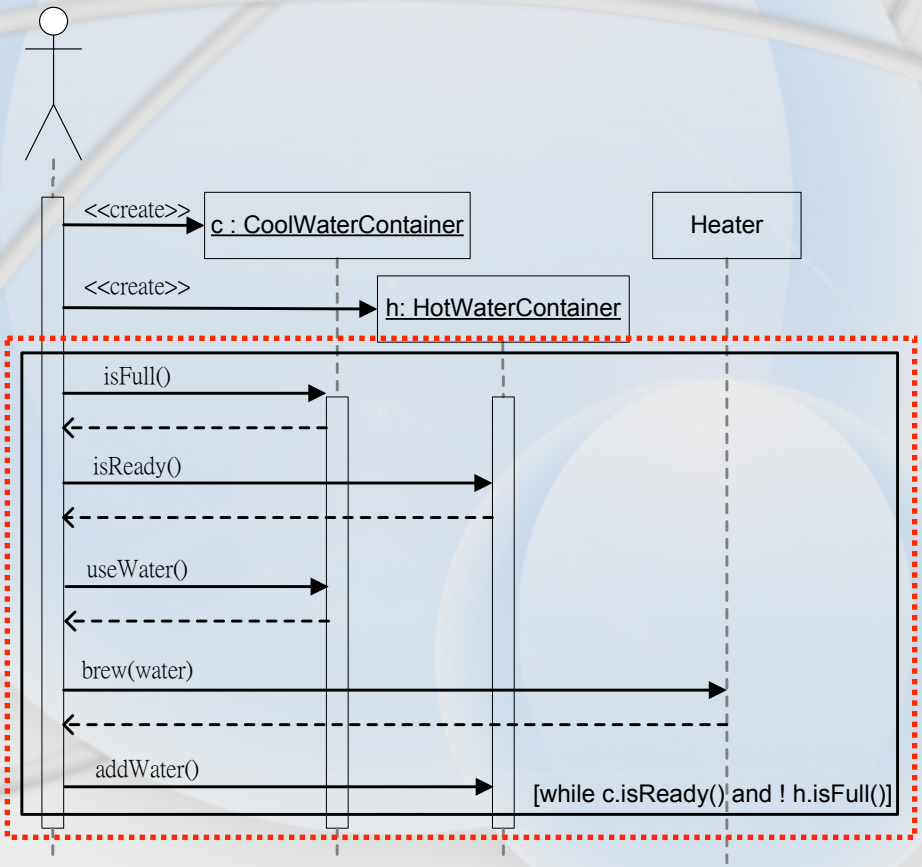
內部訊息



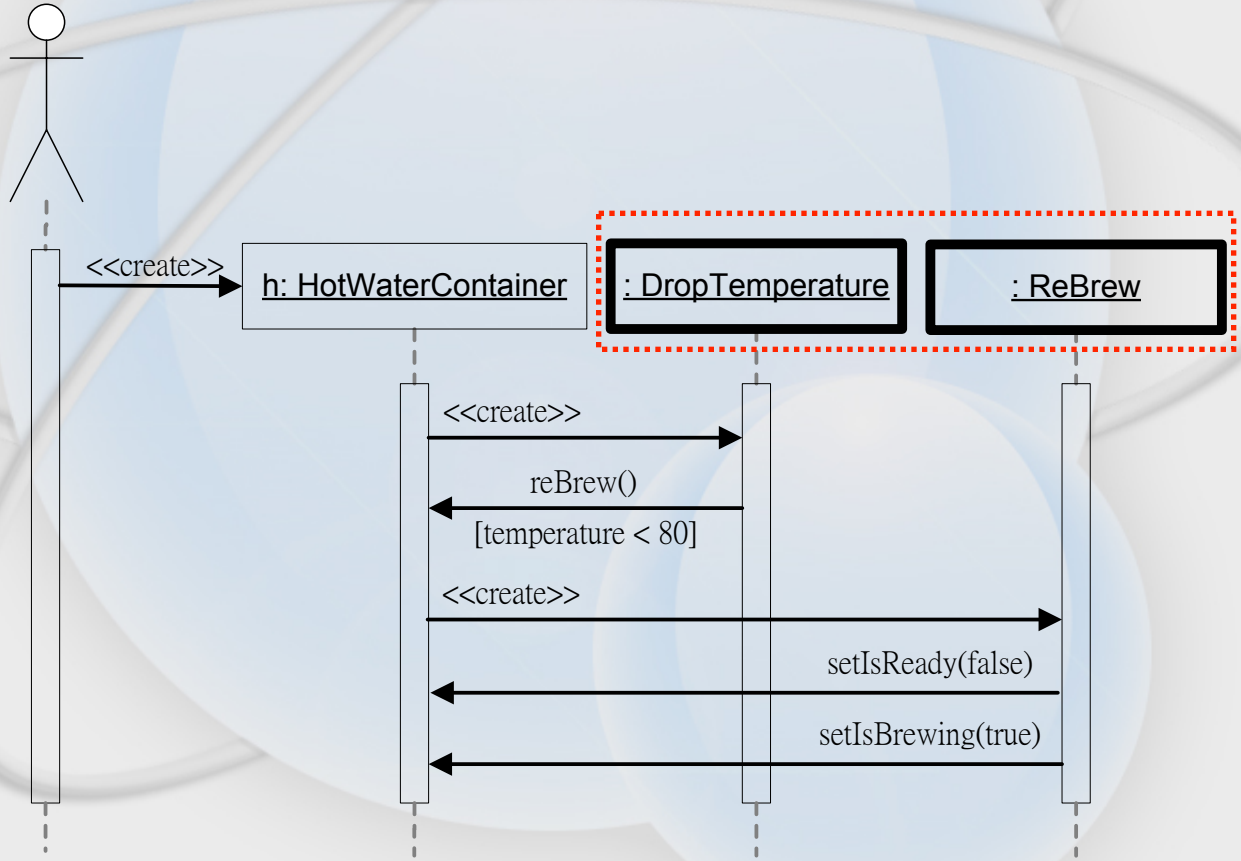
解構物件



迴圈


<<create>>
c : CoolWaterContainer
Heater
<<create>>
h: HotWaterContainer
isFull()
isReady()
useWater()
brew(water)
addWater()
[while c.isReady() and ! h.isFull()]

多執行緒


<<create>>
h: HotWaterContainer
: DropTemperature
: ReBrew
<<create>>
reBrew()
[temperature < 80]
<<create>>
setIsReady(false)
setIsBrewing(true)

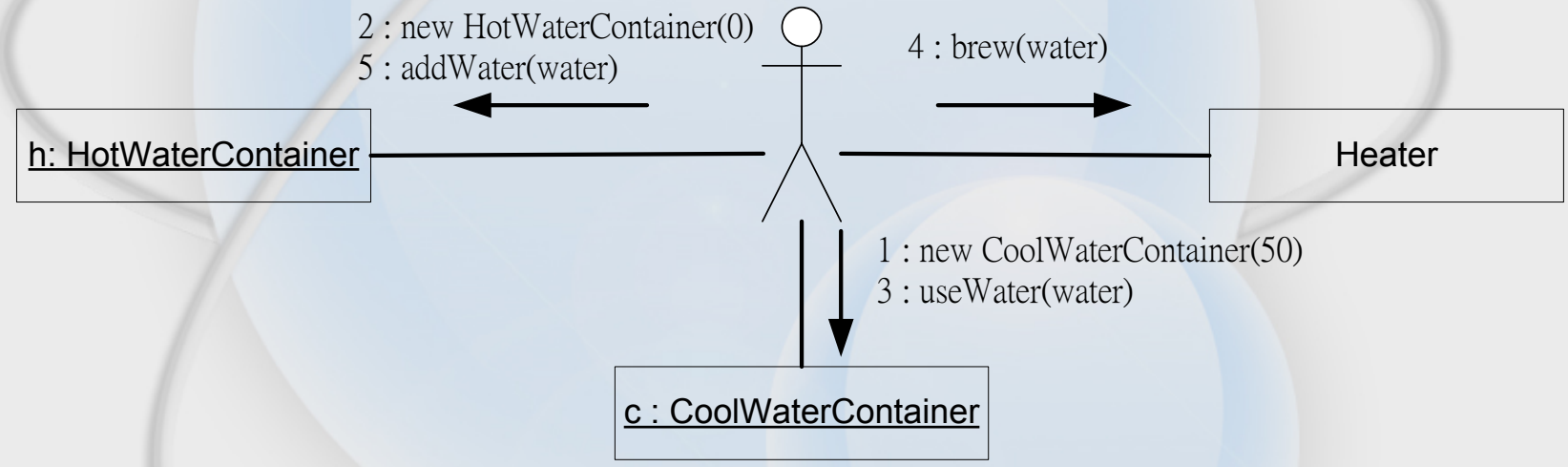
第八章

合作圖型

本章重點

- 元素
- 多執行緒

合作圖型


2 : new HotWaterContainer(0)
5 : addWater(water)
4 : brew(water)
h: HotWaterContainer
Heater
1 : new CoolWaterContainer(50)
3 : useWater(water)
c : CoolWaterContainer

元素

- 物件節點(Object node)
- 連結(Link)
- 訊息(Message)

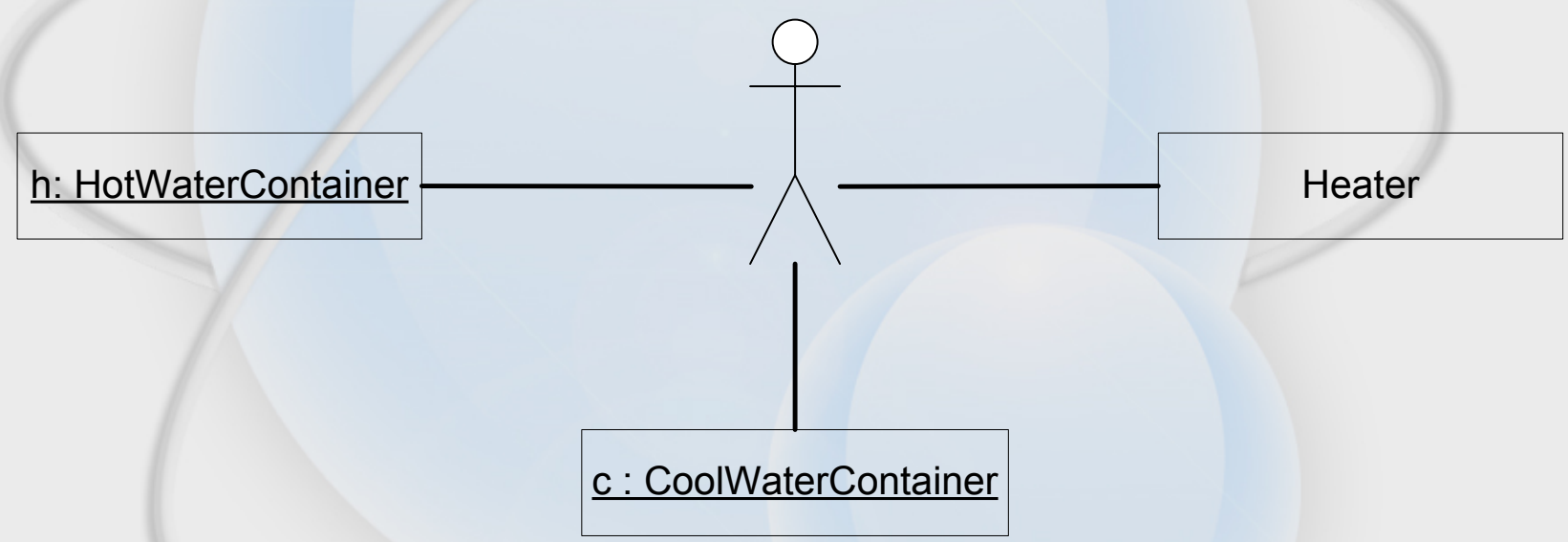
物件節點

h: HotWaterContainer

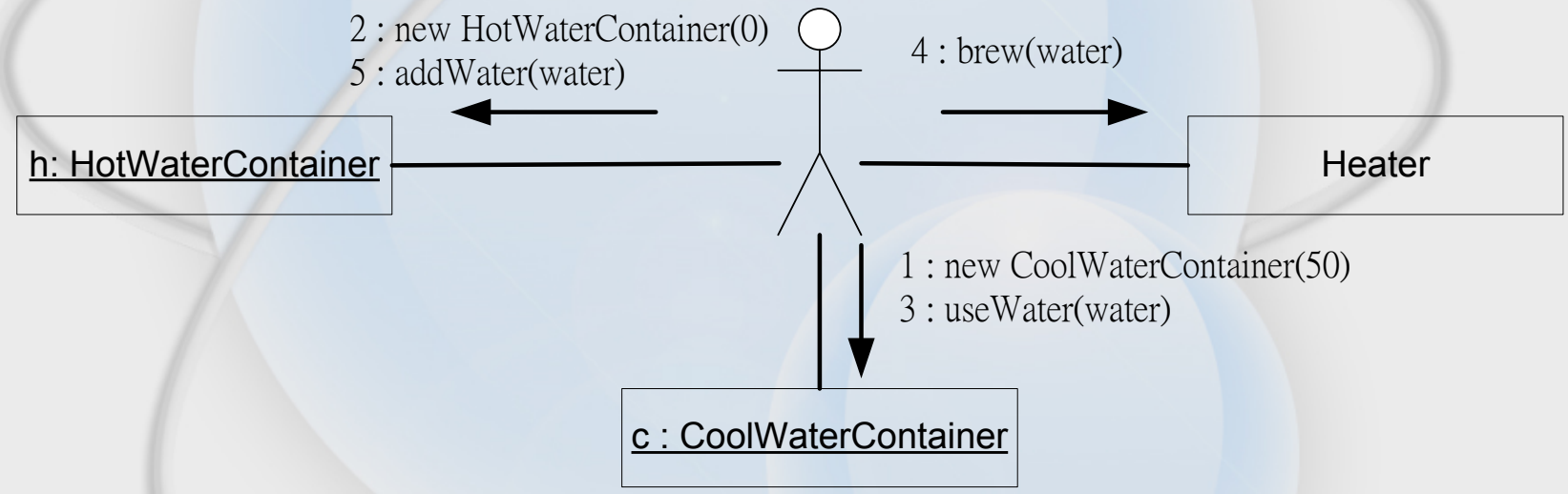
c : CoolWaterContainer

Heater

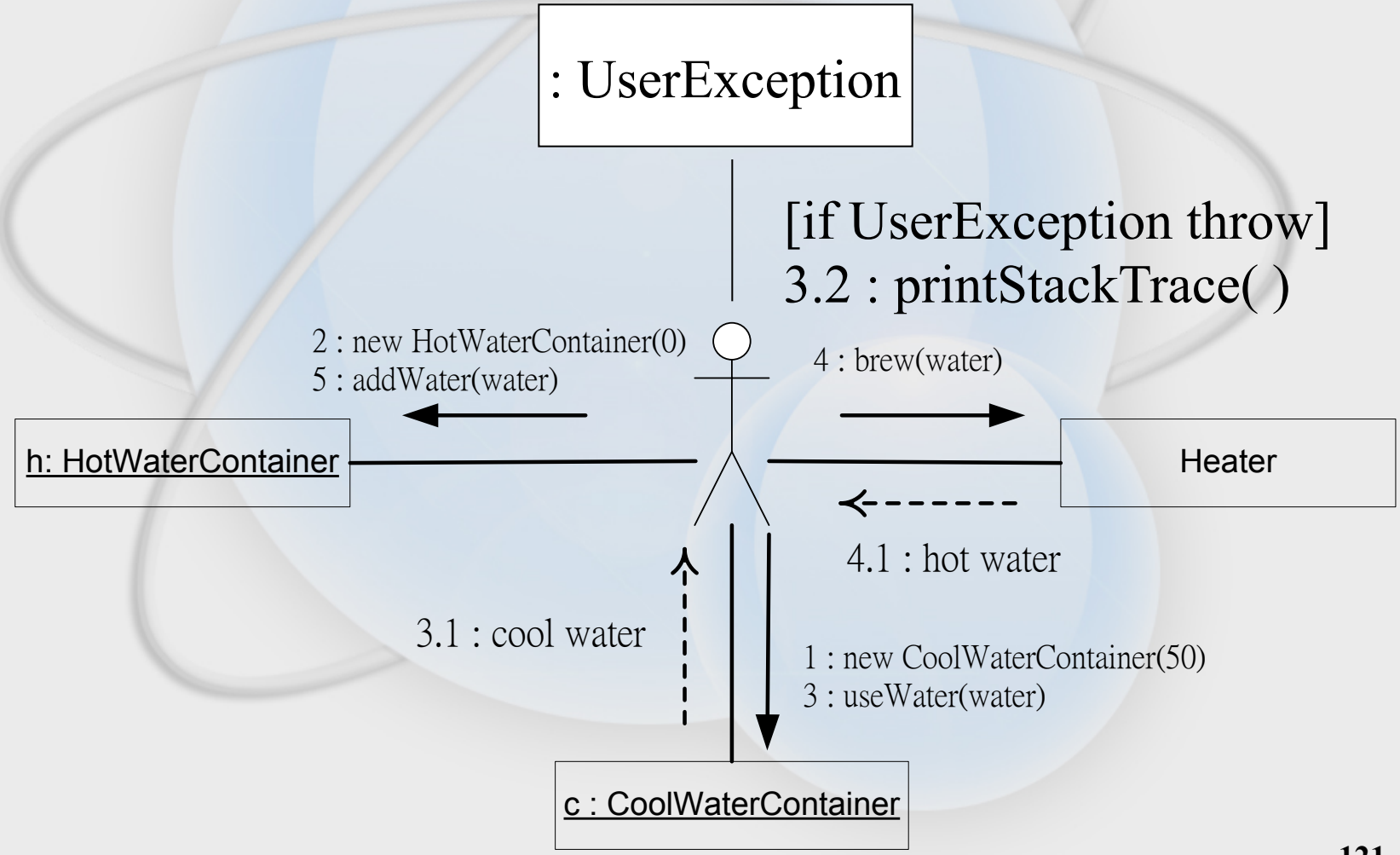
連結



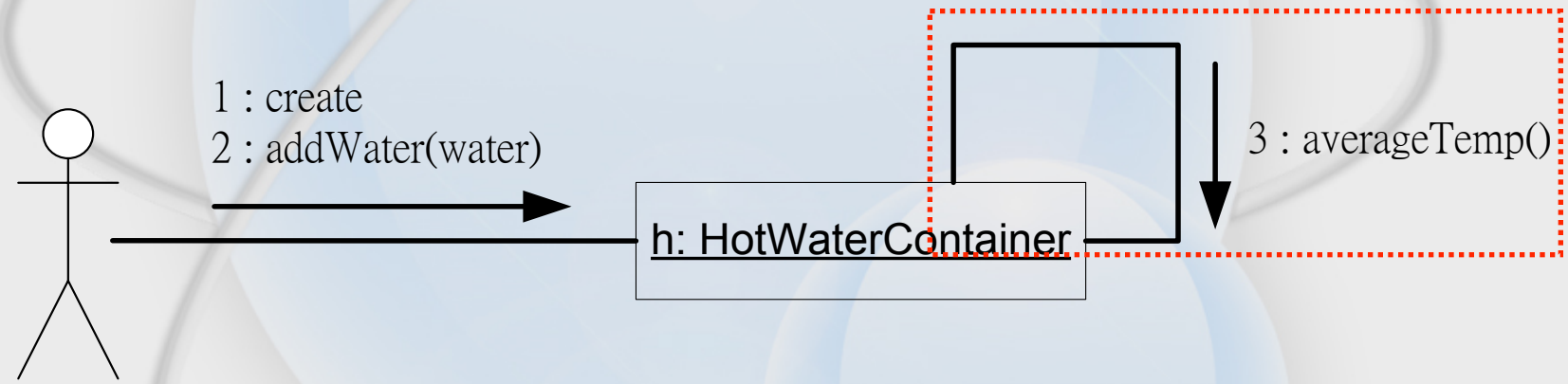
訊息


2 : new HotWaterContainer(0)
5 : addWater(water)
4 : brew(water)
h: HotWaterContainer
Heater
1 : new CoolWaterContainer(50)
3 : useWater(water)
c : CoolWaterContainer

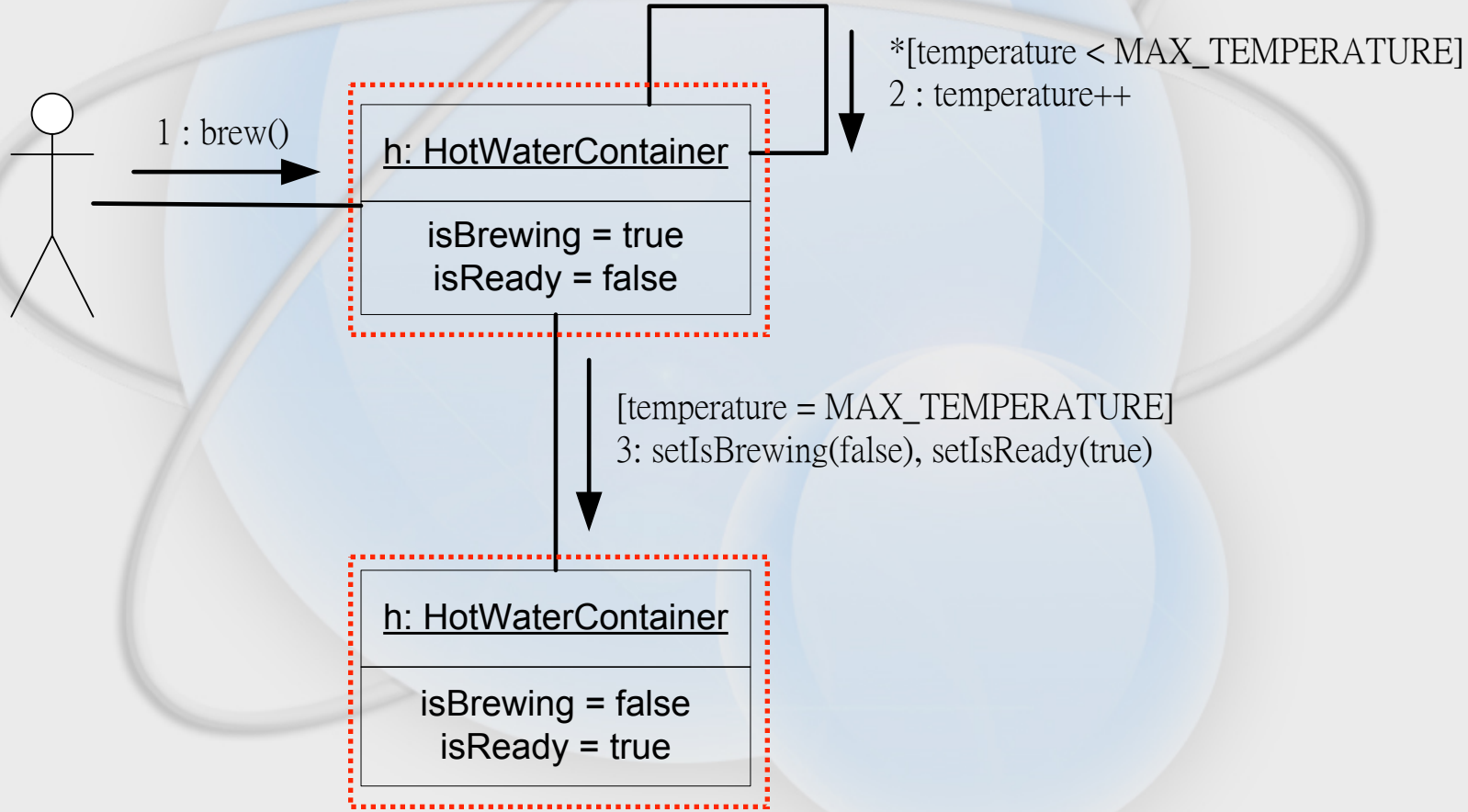
訊息的回傳值


: UserException
[if UserException throw]
3.2 : printStackTrace( )
2 : new HotWaterContainer(0)
5 : addWater(water)
4 : brew(water)
h: HotWaterContainer
Heater
4.1 : hot water
3.1 : cool water
1 : new CoolWaterContainer(50)
3 : useWater(water)
c : CoolWaterContainer

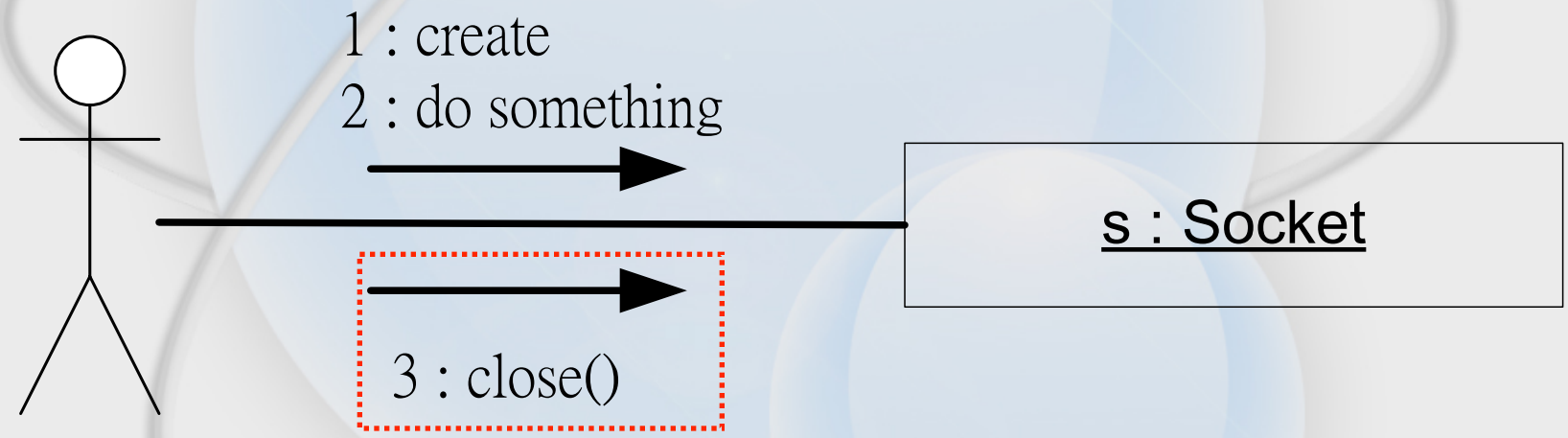
內部訊息


1 : create
2 : addWater(water)
h: HotWaterContainer
3 : averageTemp()

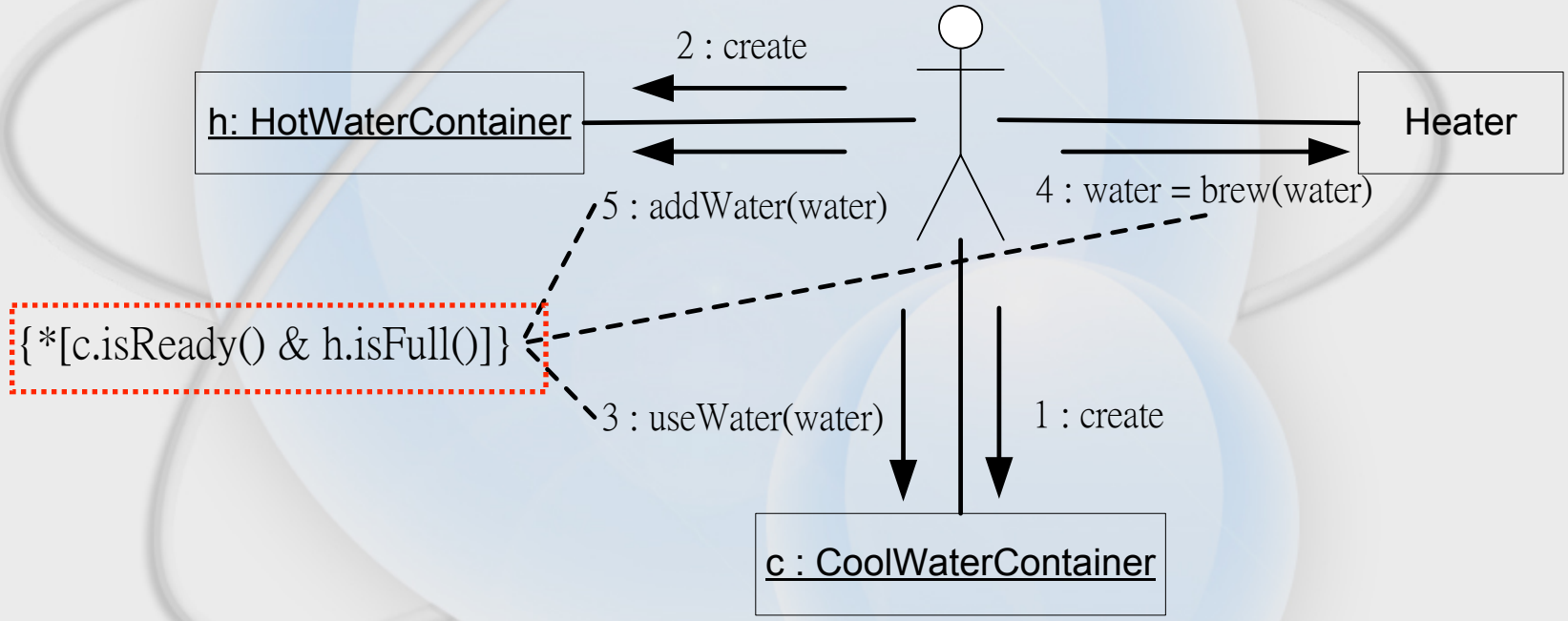
物件狀態的改變


1 : brew()
h: HotWaterContainer
isBrewing = true
isReady = false
*[temperature < MAX_TEMPERATURE]
2 : temperature++
[temperature = MAX_TEMPERATURE]
3: setIsBrewing(false), setIsReady(true)
h: HotWaterContainer
isBrewing = false
isReady = true

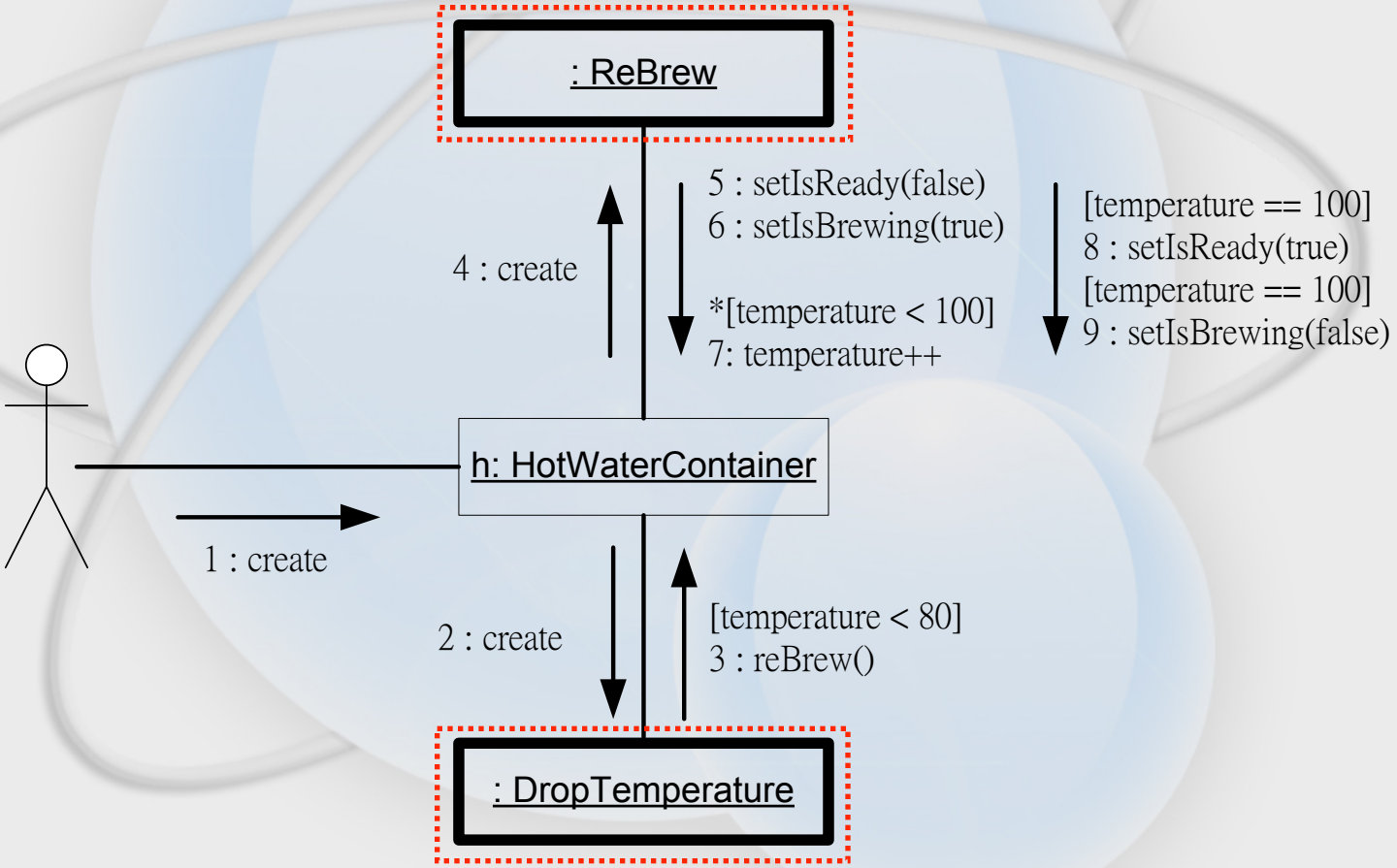
解構物件



迴圈


2 : create
h: HotWaterContainer
Heater
5 : addWater(water)
4 : water = brew(water)
{*[c.isReady() & h.isFull()]}
3 : useWater(water)
1 : create
c : CoolWaterContainer

多執行緒


: ReBrew
5 : setIsReady(false)
6 : setIsBrewing(true)
4 : create
*[temperature < 100]
7: temperature++
[temperature == 100]
8 : setIsReady(true)
[temperature == 100]
9 : setIsBrewing(false)
h: HotWaterContainer
1 : create
2 : create
[temperature < 80]
3 : reBrew()
: DropTemperature

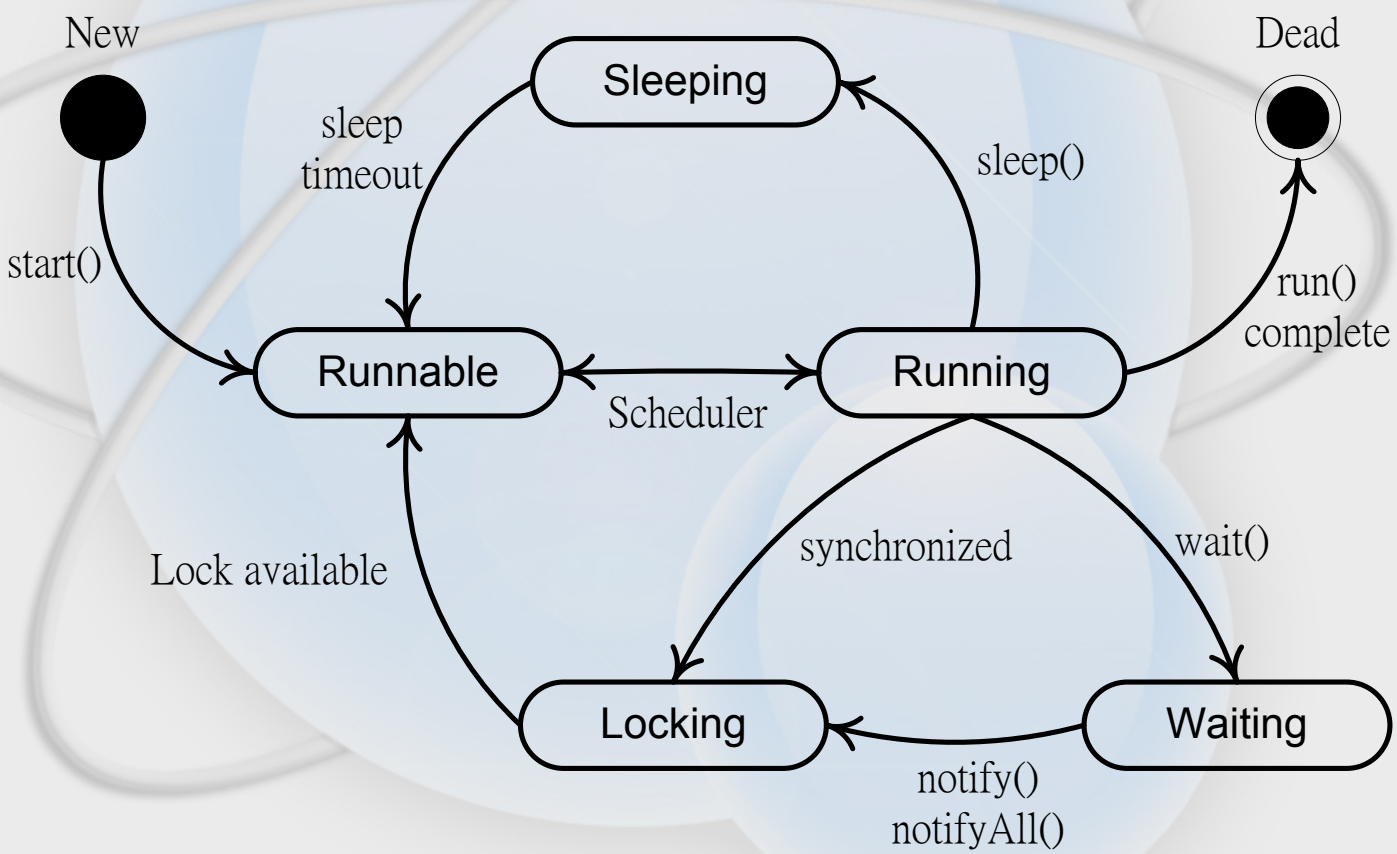
第九章

狀態圖型

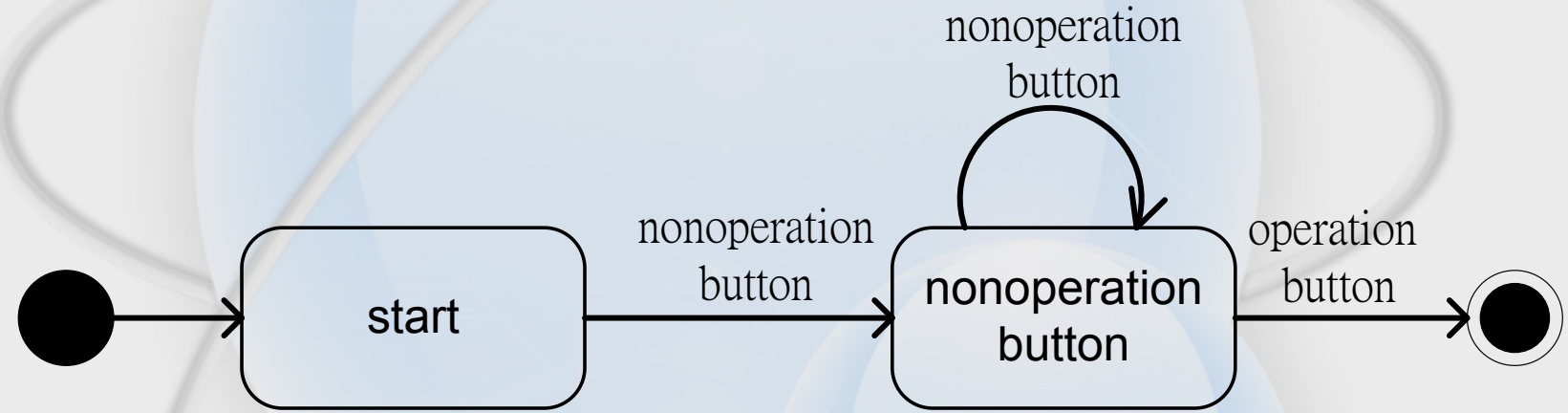
本章重點

- 狀態節點
- 轉換
- 子狀態

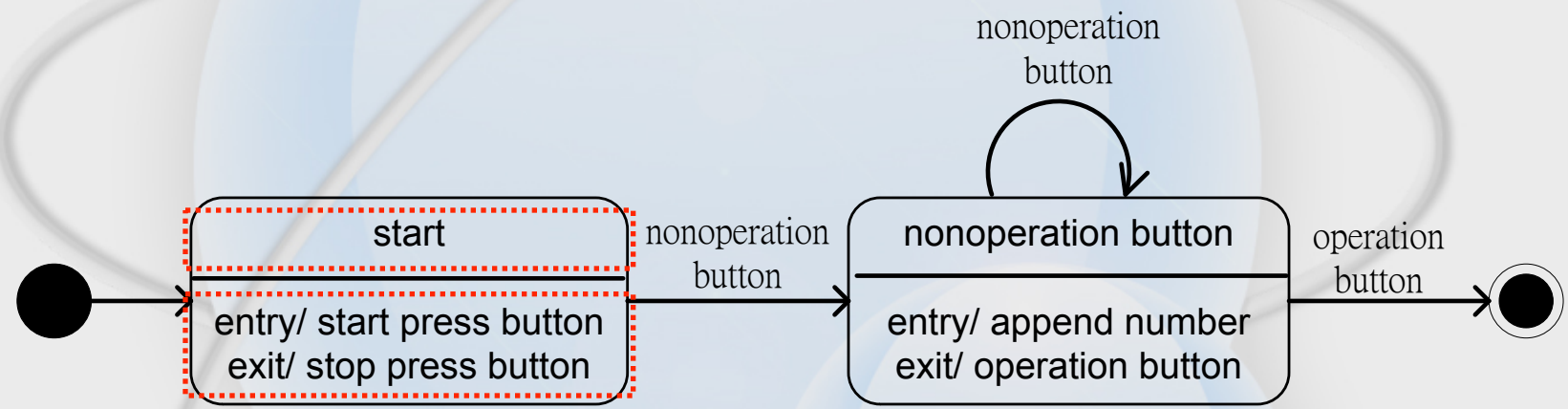
狀態圖型


New
Dead
Sleeping
sleep
timeout
sleep()
start()
run()
complete
Runnable
Running
Scheduler
synchronized
wait()
Lock available
Locking
Waiting
notify()
notifyAll()

狀態節點


nonoperation
button
nonoperation
button
operation
button
start
nonoperation
button

內部轉換區格


nonoperation
button
start
entry/ start press button
exit/ stop press button
nonoperation
button
nonoperation button
entry/ append number
exit/ operation button
operation
button

內部轉換區格

標籤	說明
entry	進入狀態節點時的動作
exit	離開狀態節點時的動作
do	停留在這個狀態節點時執行的動作

<事件名稱>(<參數>) / <動作>

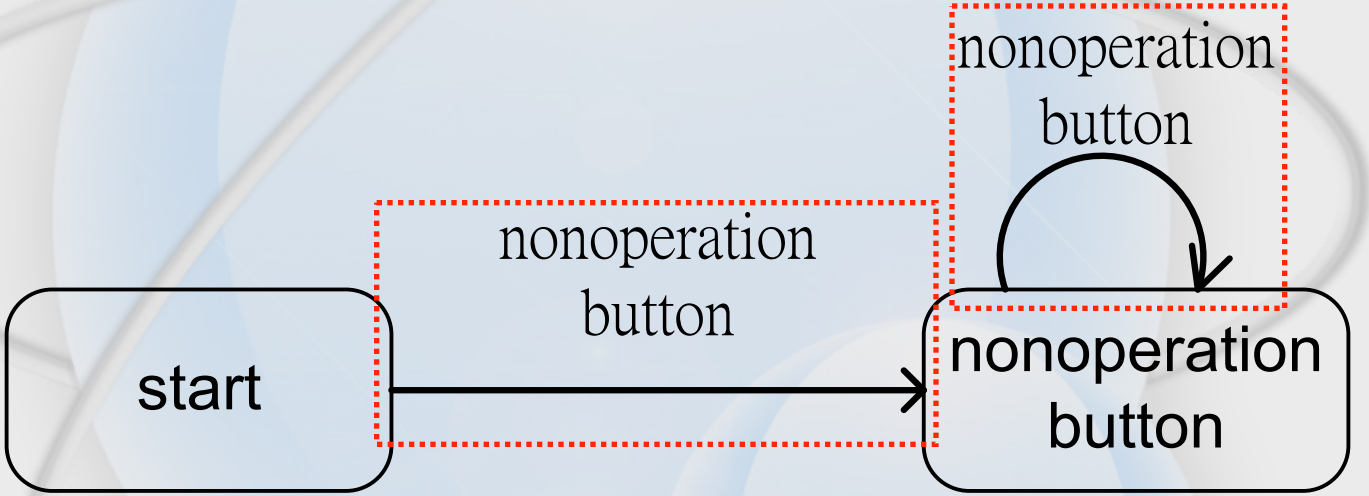
內部轉換區格

typing password

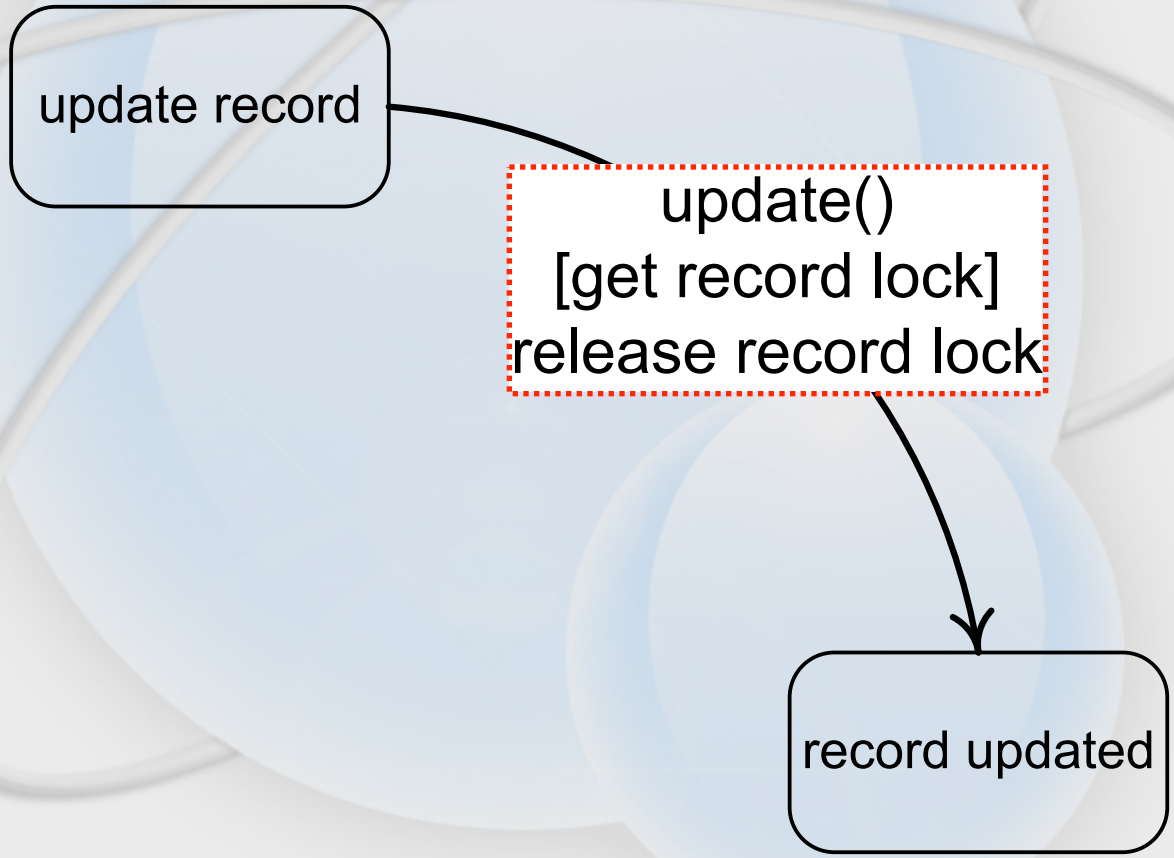
---

entry/ set response to *
exit/ set response to normal
charcter(c)/ process character

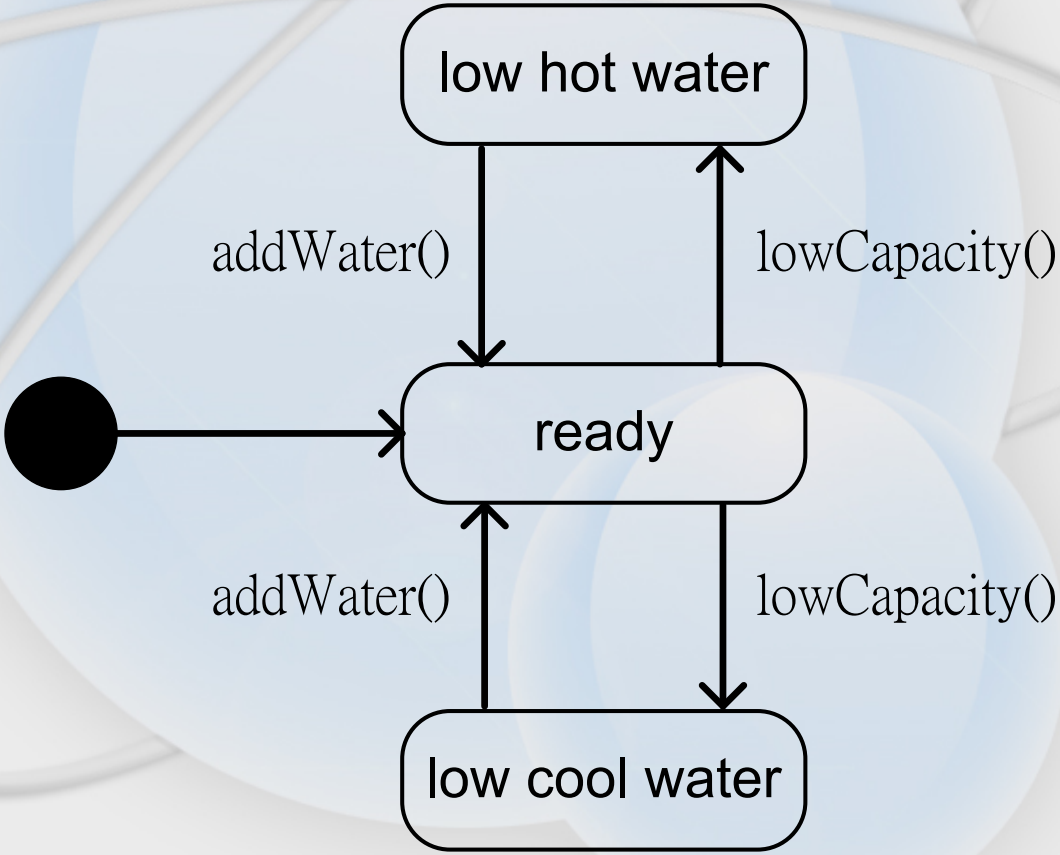
轉換


nonoperation
button
nonoperation
button
start
nonoperation
button

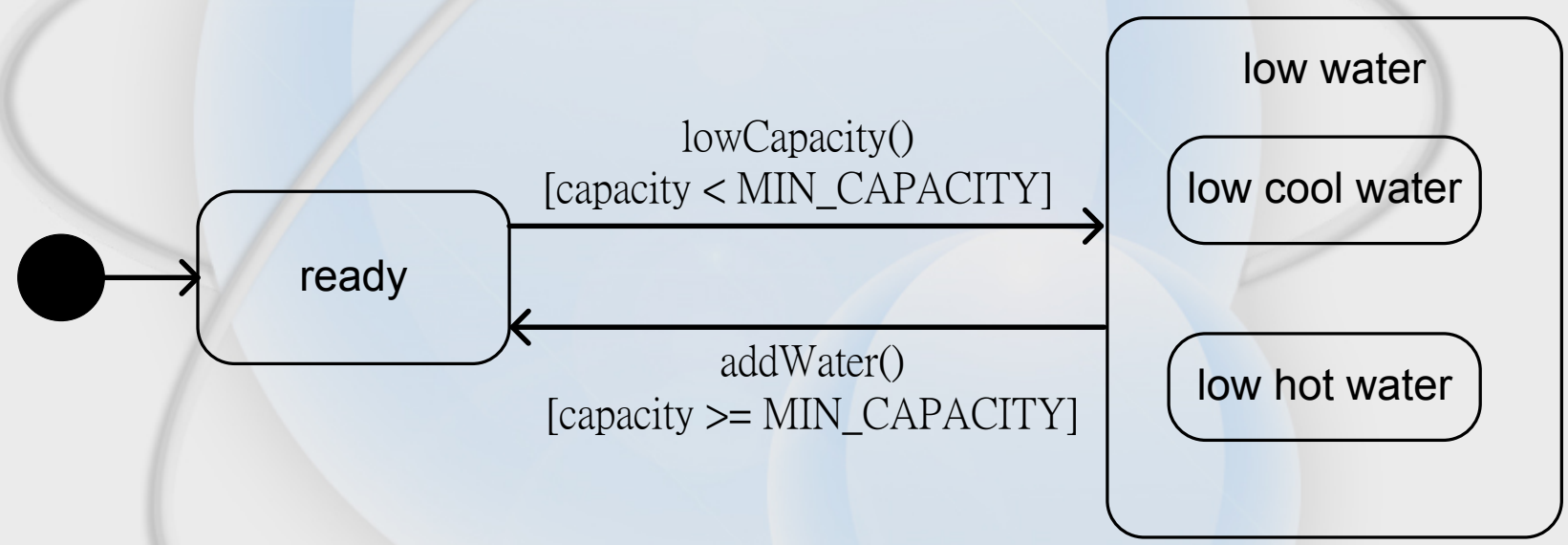
事件


update record
update()
[get record lock]
release record lock
record updated

子狀態


low hot water
addWater()
lowCapacity()
ready
addWater()
lowCapacity()
low cool water

子狀態


ready
lowCapacity()
[capacity < MIN_CAPACITY]
addWater()
[capacity >= MIN_CAPACITY]
low water
low cool water
low hot water

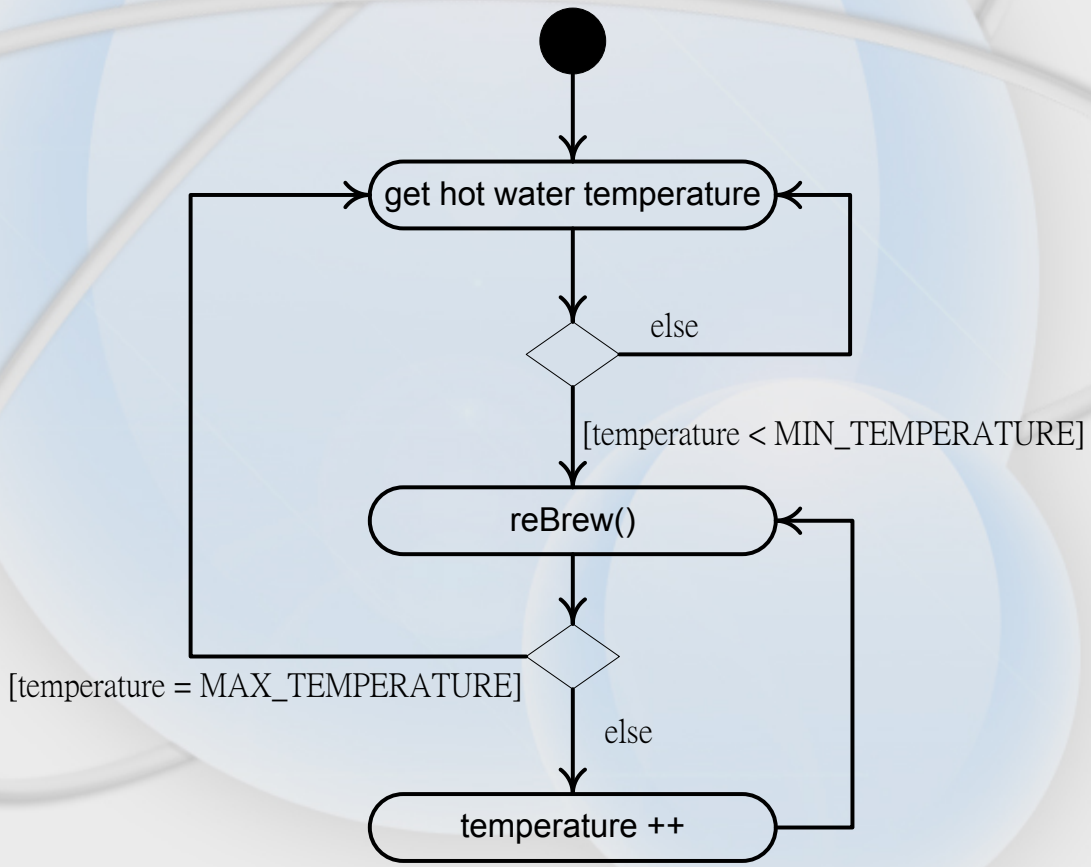
第十章

活動圖型

本章重點

- 圖型元素

活動圖型


get hot water temperature
else
[temperature < MIN_TEMPERATURE]
reBrew()
[temperature = MAX_TEMPERATURE]
else
temperature ++

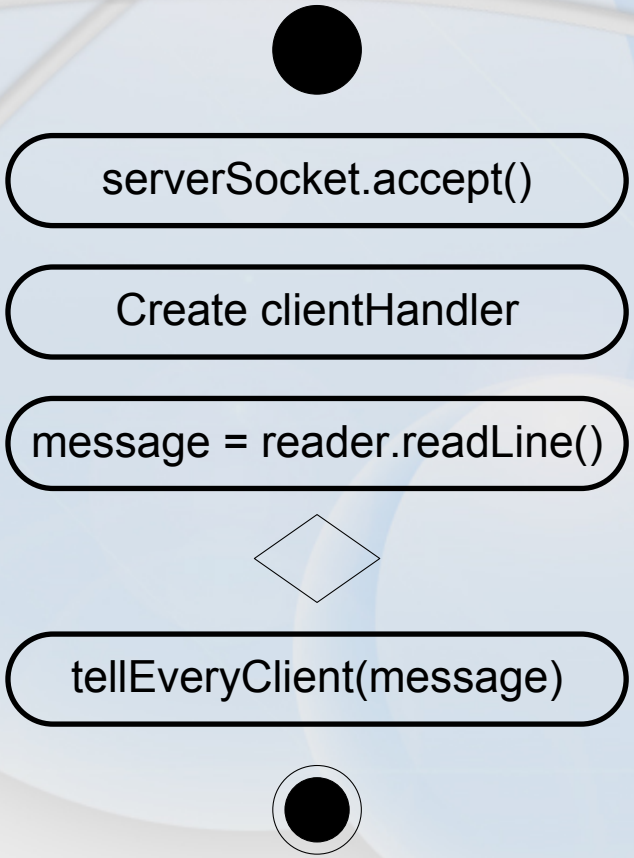
主要的用途

- 針對循序圖型中比較複雜的訊息傳遞加以說明
- 顯示在狀態圖型裡比較複雜的轉換事件
- 說明合作圖型中的訊息

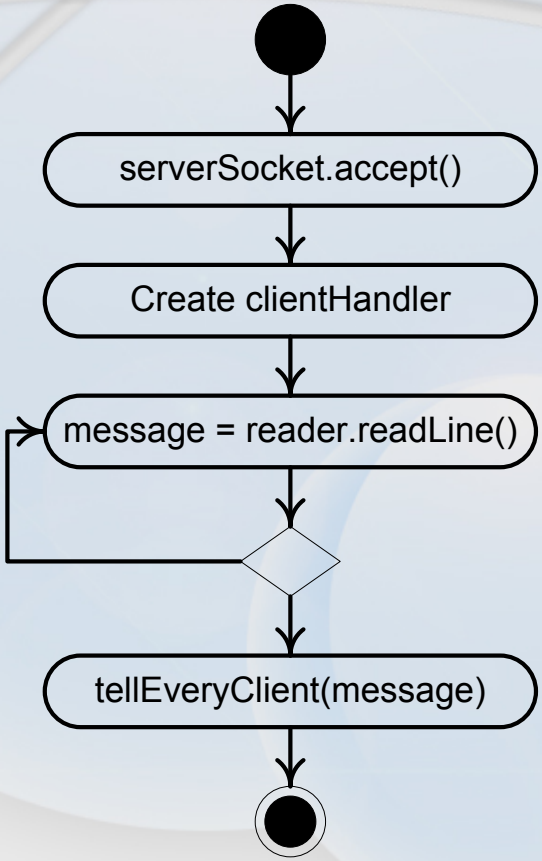
圖型元素

- 狀態與活動(State and Activity)
- 轉換(Transition)
- 分支(Branch)
- 分岐與結合(Fork and Join)
- 水道(Swimlane)

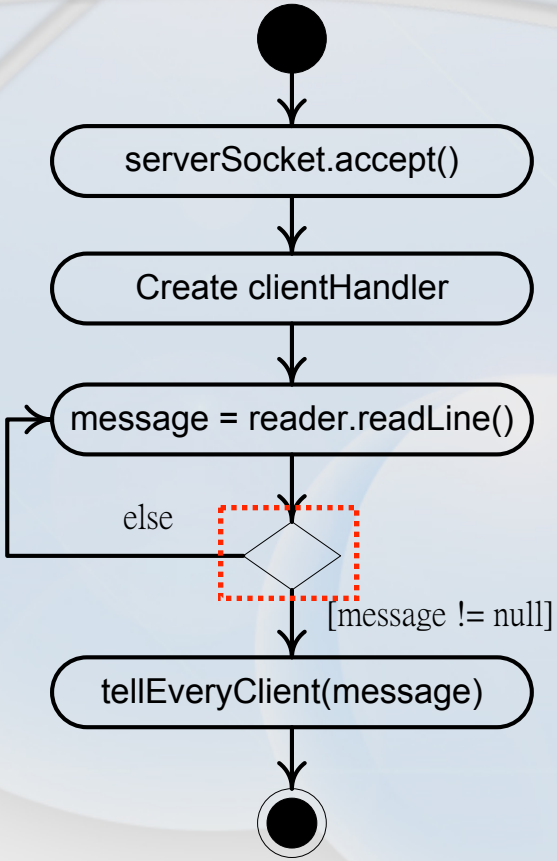
狀態與活動


serverSocket.accept()
Create clientHandler
message = reader.readLine()
tellEveryClient(message)

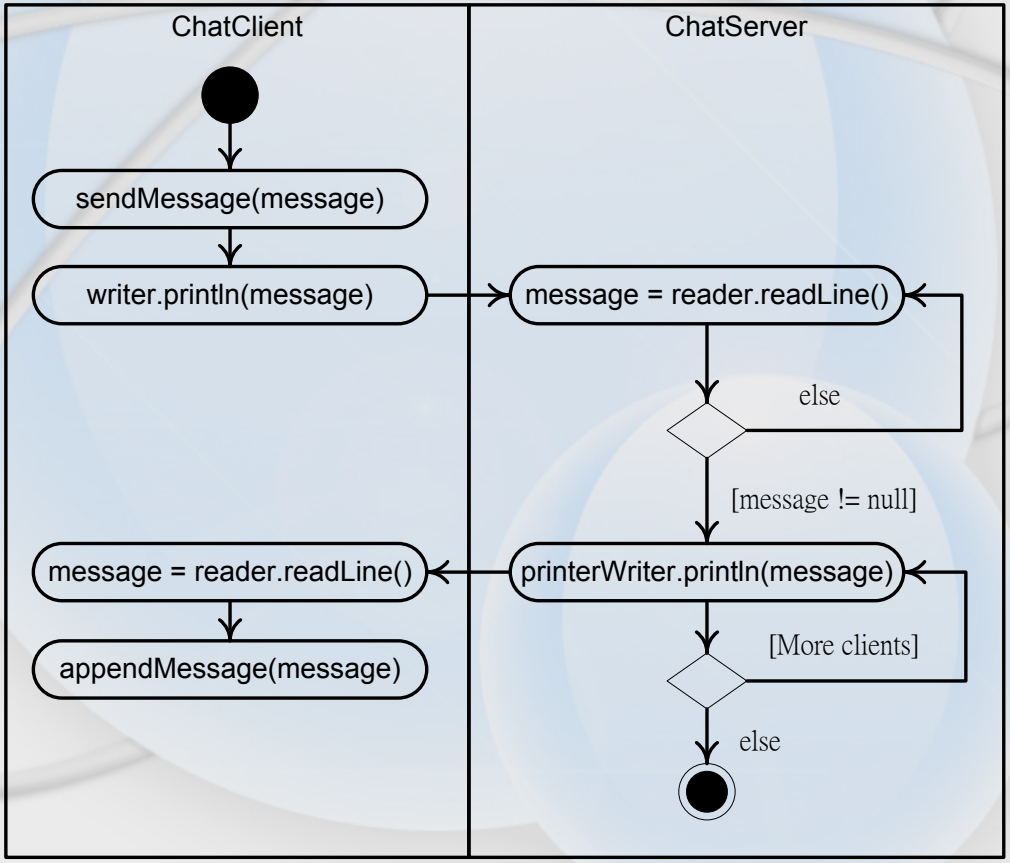
轉換


serverSocket.accept()
Create clientHandler
message = reader.readLine()
tellEveryClient(message)

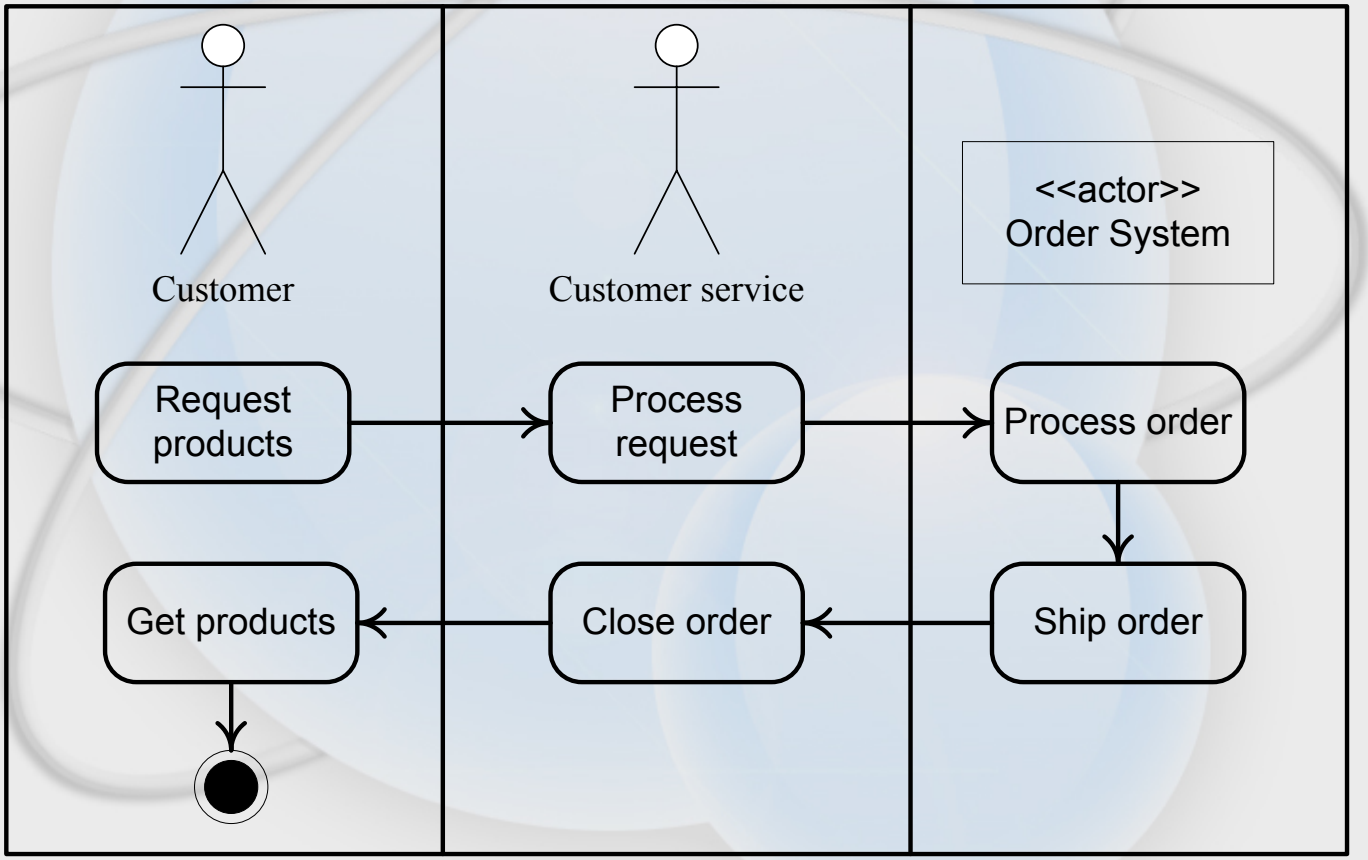
分支


serverSocket.accept()
Create clientHandler
message = reader.readLine()
else
[message != null]
tellEveryClient(message)

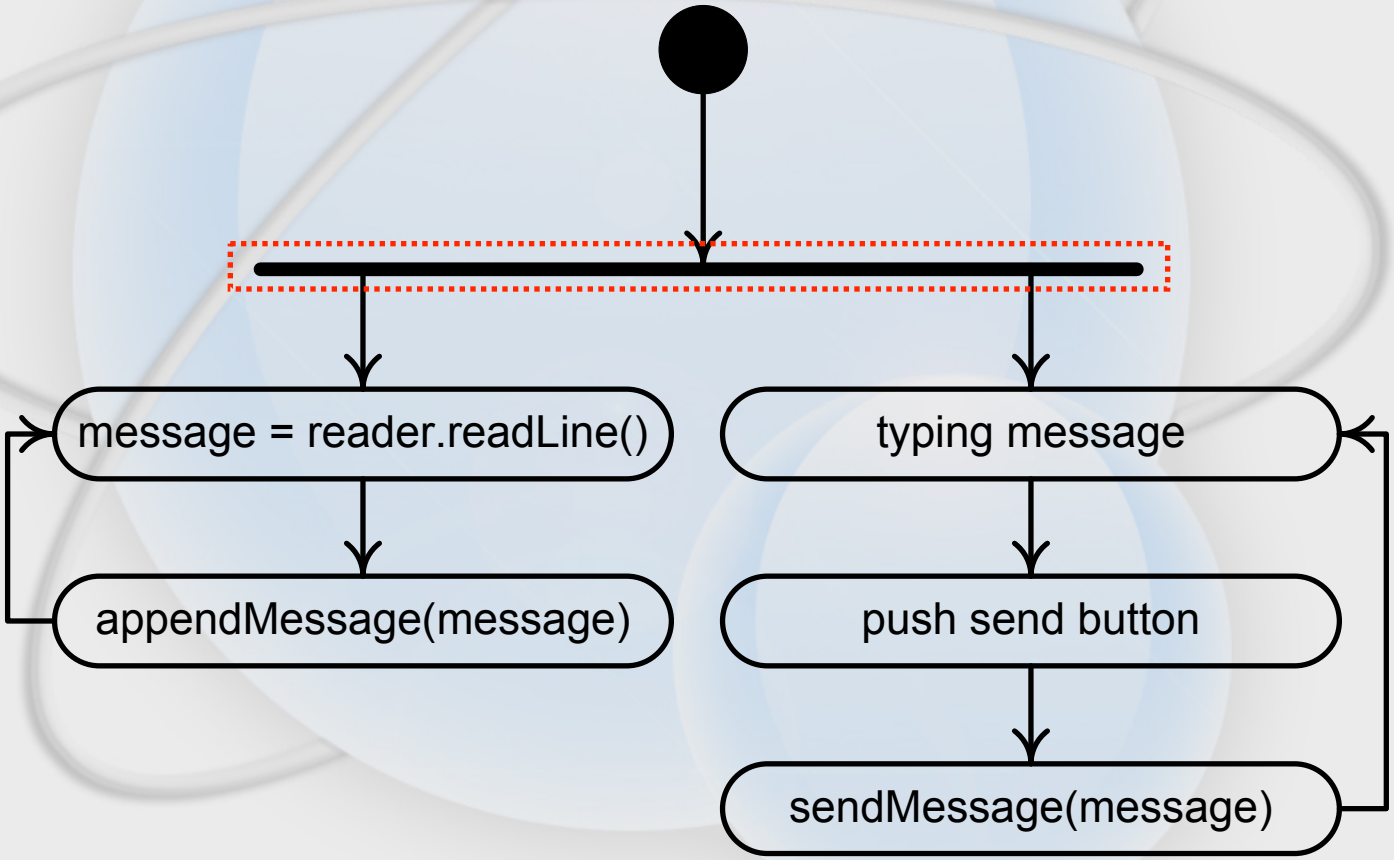
水道


ChatClient
ChatServer
sendMessage(message)
writer.println(message)
message = reader.readLine()
else
[message != null]
message = reader.readLine()
printerWriter.println(message)
appendMessage(message)
[More clients]
else

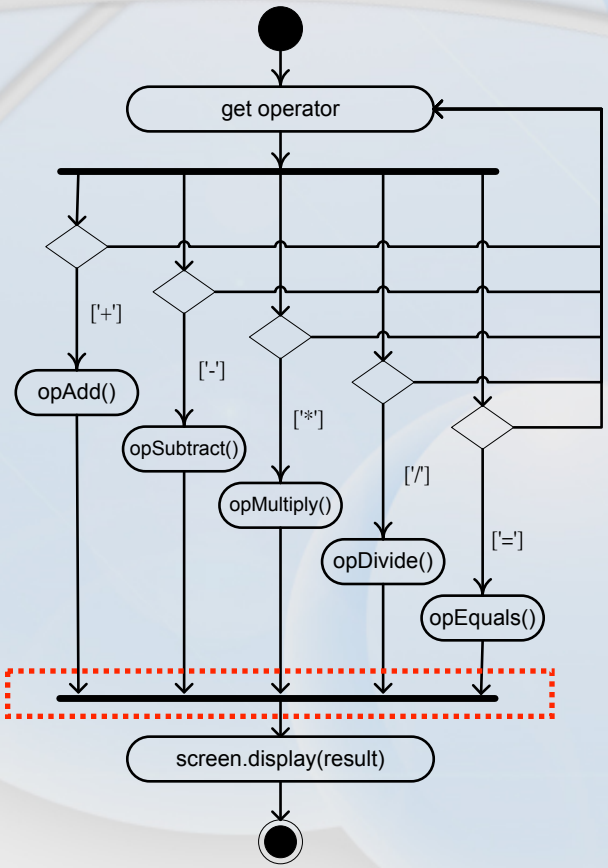
水道


Customer
Customer service
<<actor>>
Order System
Request products
Process request
Process order
Get products
Close order
Ship order

分岐


message = reader.readLine()
appendMessage(message)
typing message
push send button
sendMessage(message)

結合


get operator
['+']
['-']
['*']
['/']
['=']
opAdd()
opSubtract()
opMultiply()
opDivide()
opEquals()
screen.display(result)