

# Builder Pattern

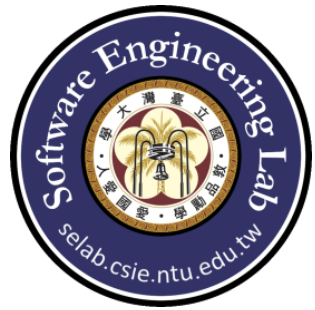
Prof. Jonathan Lee (李允中)  
Department of CSIE  
National Taiwan University



# Design Aspect of Builder

---

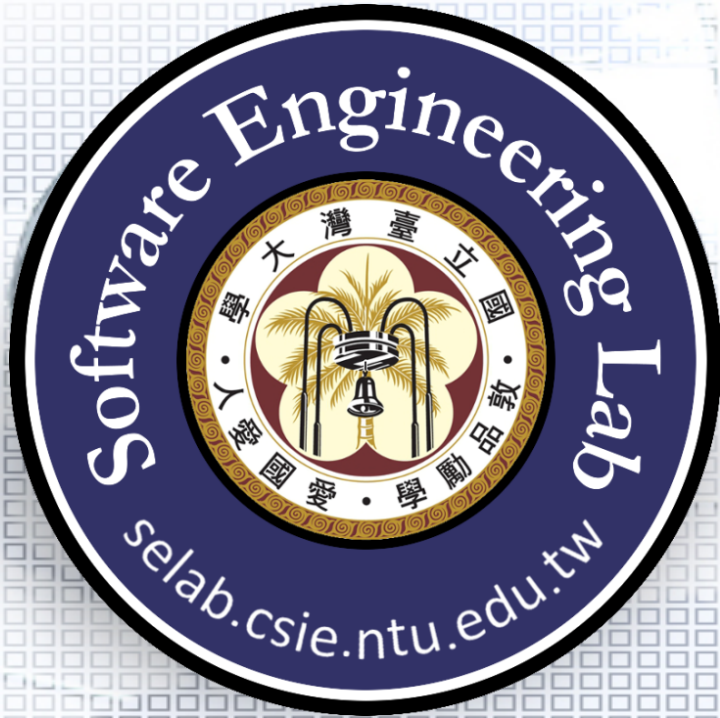
how a composite object gets created



# Outline

---

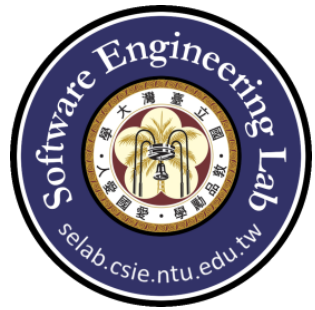
- ☐ An RTF (Rich Text Format) Converter Requirements Statements
- ☐ Initial Design and Its Problems
- ☐ Design Process
- ☐ Refactored Design after Design Process
- ☐ Recurrent Problems
- ☐ Intent
- ☐ Builder Pattern Structure
- ☐ Abstract Factory vs. Builder
- ☐ A Vacation Planner: Another Example
- ☐ Homework



# A Rich Text Format Converter (Builder)

Prof. Jonathan Lee (李允中)

Department of Computer Science and  
Information Engineering  
National Taiwan University



# Homework: Requirements Statement

---

- ☐ A reader for the RTF (Rich Text Format) document exchange format should be able to convert RTF to many text formats.
- ☐ The reader converts RTF documents into TeX text or into a text widget by recognizing different RTF tokens (Character, Font Change and Paragraph).



# Requirements Statements<sub>1</sub>

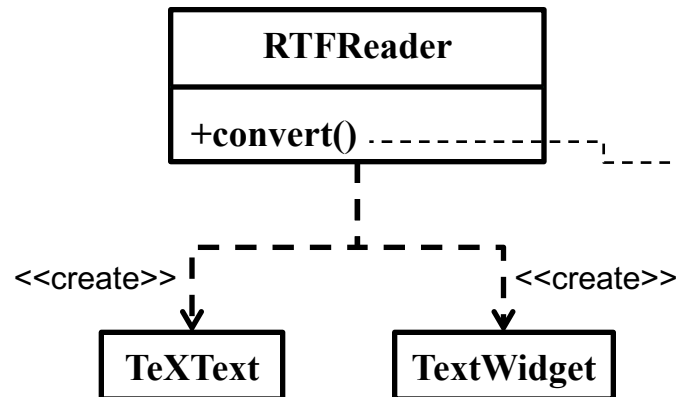
- ❑ A reader for the RTF (Rich Text Format) document exchange format should be able to convert RTF to many text formats.

<b>RTFReader</b>
<b>+convert()</b>



# Requirements Statements<sub>2</sub>

- ❑ The reader converts RTF documents into TeX text or into a text widget by recognizing different RTF tokens (Character, Font Change and Paragraph).

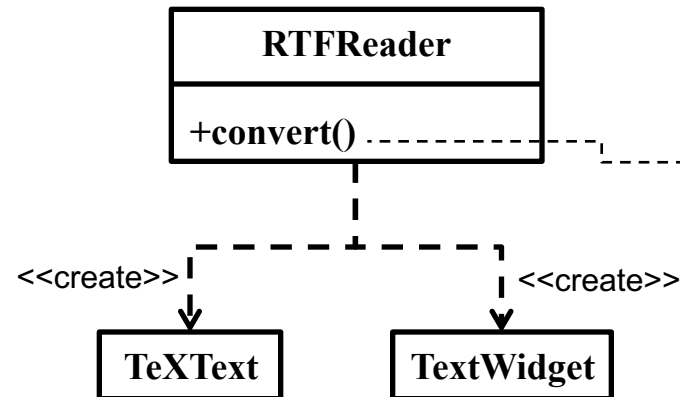


```
if(converterType=="TeX"){
  for(Character token : content.toCharArray()){
    switch(token){
      case CHAR:
        //Convert Character Step
      case FONT:
        //Convert Font Change Step
      case PARA:
        //Convert Paragraph Step
    }
  }
  // return the TeX document
}else{
  for(Character token : content.toCharArray()){
    switch(token){
      case CHAR:
        //Convert Character Step
      case FONT:
        //Convert Font Change Step
      case PARA:
        //Convert Paragraph Step
    }
  }
  // return the TextWidget document
}
```





# Initial Design - Class Diagram

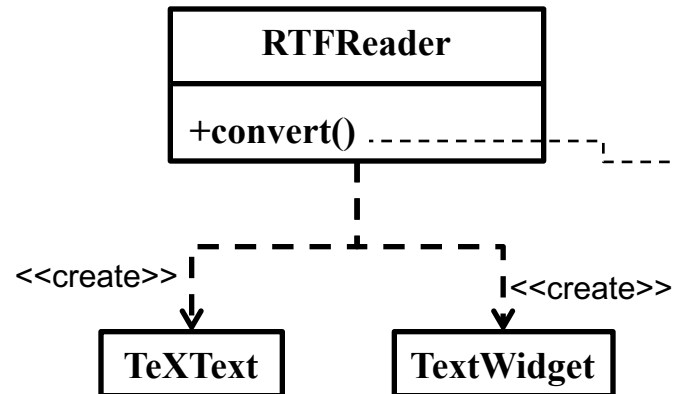


```
if(converterType=="TeX"){
for(Character token : content.toCharArray()){
    switch(token){
        case CHAR:
            //Convert Character Step
        case FONT:
            //Convert Font Change Step
        case PARA:
            //Convert Paragraph Step
    }
}
// return the TeX document
}else{
for(Character token : content.toCharArray()){
    switch(token){
        case CHAR:
            //Convert Character Step
        case FONT:
            //Convert Font Change Step
        case PARA:
            //Convert Paragraph Step
    }
}
// return the TextWidget document
}
```



# Problems with Initial Design

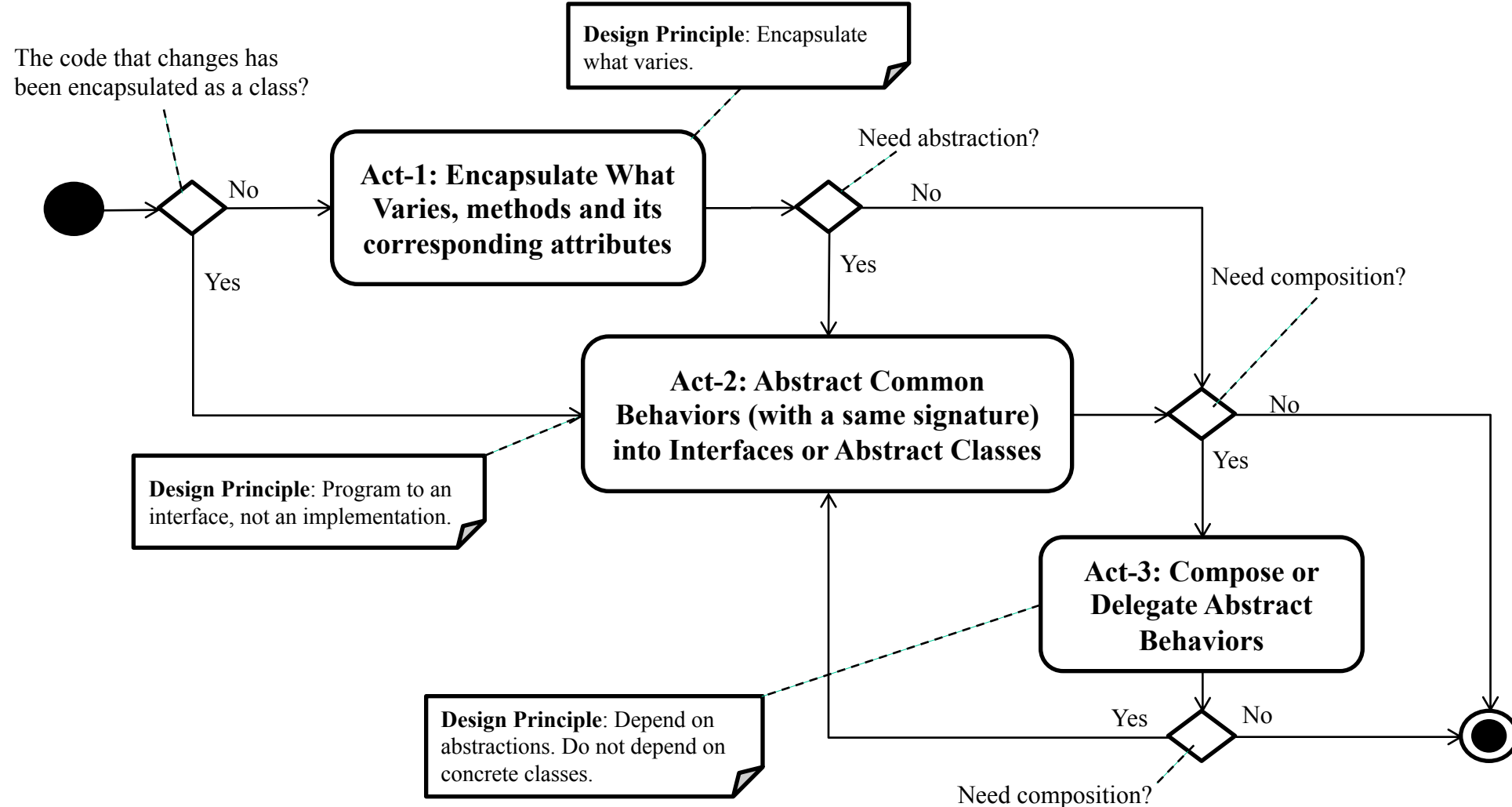
Problem: When adding a new conversion, we need to modify the reader.



```

if(converterType=="TeX"){
for(Character token : content.toCharArray()){
    switch(token){
        case CHAR:
            //Convert Character Step
        case FONT:
            //Convert Font Change Step
        case PARA:
            //Convert Paragraph Step
    }
}
// return the TeX document
}else{
for(Character token : content.toCharArray()){
    switch(token){
        case CHAR:
            //Convert Character Step
        case FONT:
            //Convert Font Change Step
        case PARA:
            //Convert Paragraph Step
    }
}
// return the TextWidget document
}
}
    
```

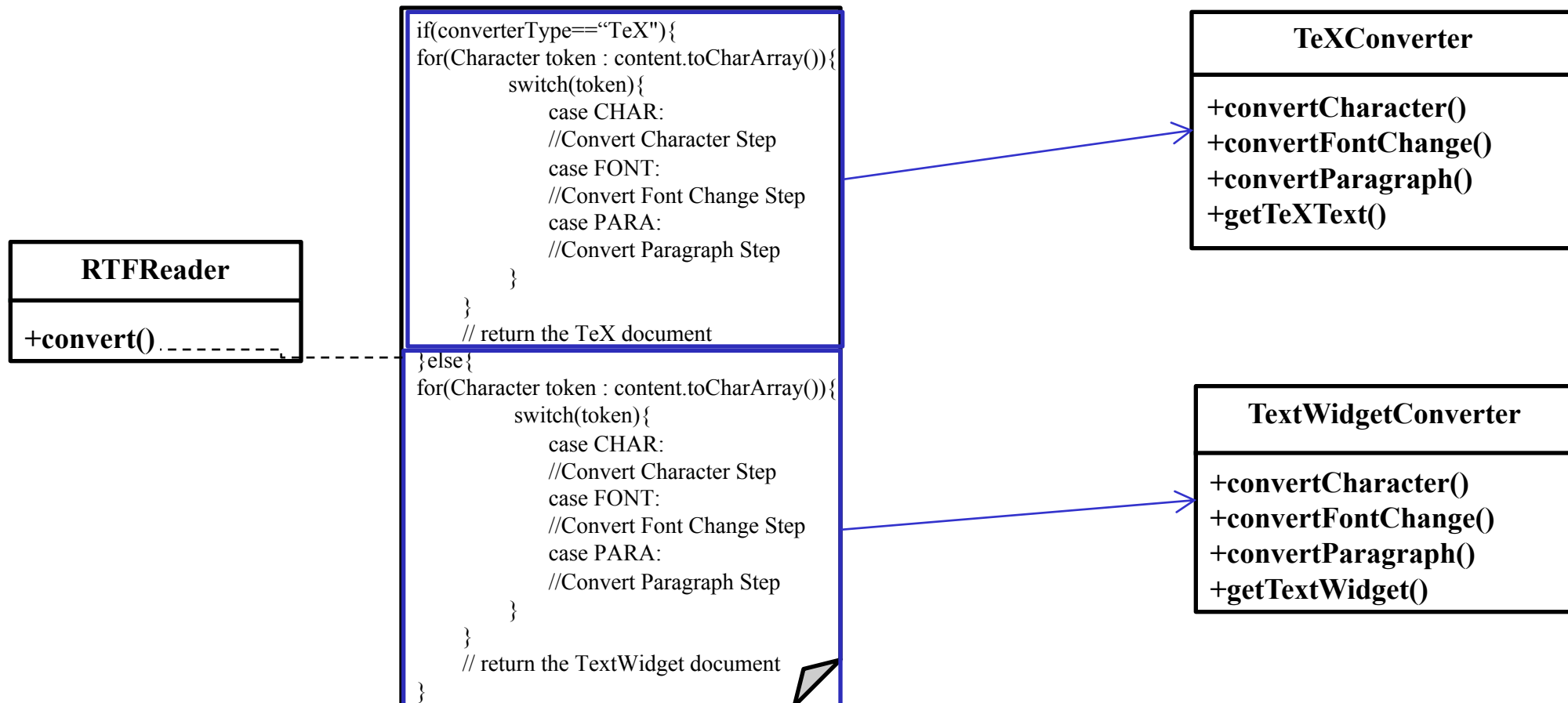
# Design Process for Change

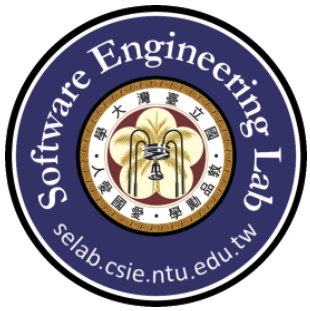




# Act-1: Encapsulate What Varies

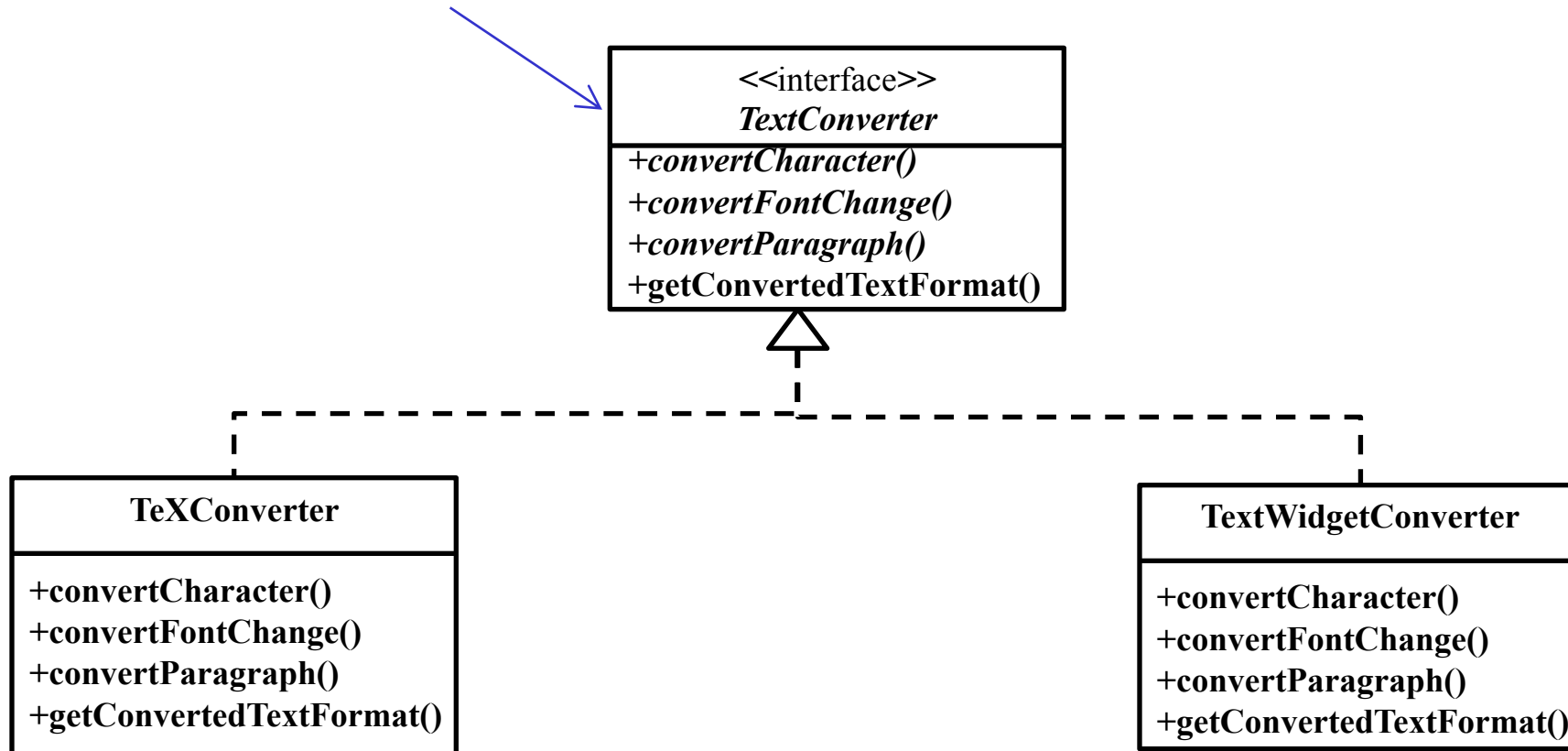
## Act-1.3: Encapsulate a part of a method body into a concrete class





# Act-2: Abstract Common Behaviors

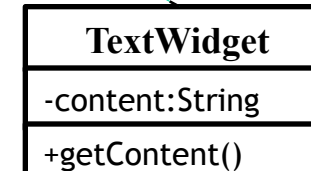
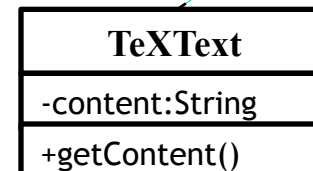
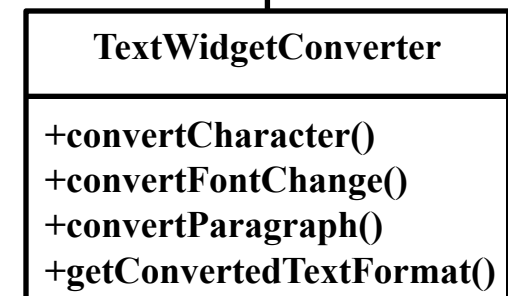
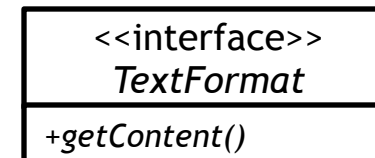
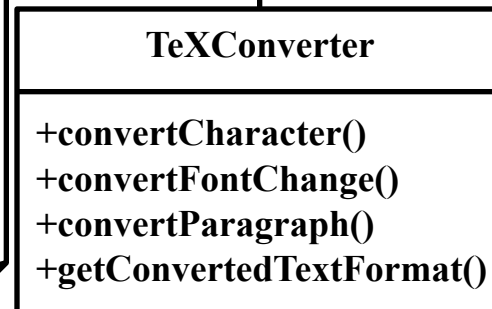
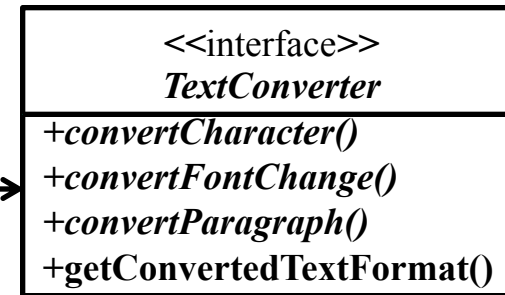
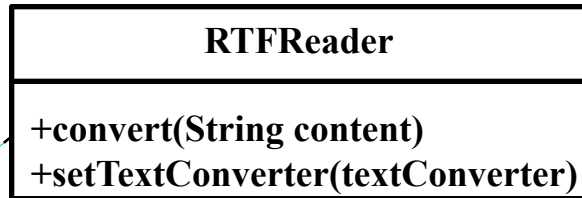
Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism





# Act-3: Compose Abstract Behaviors

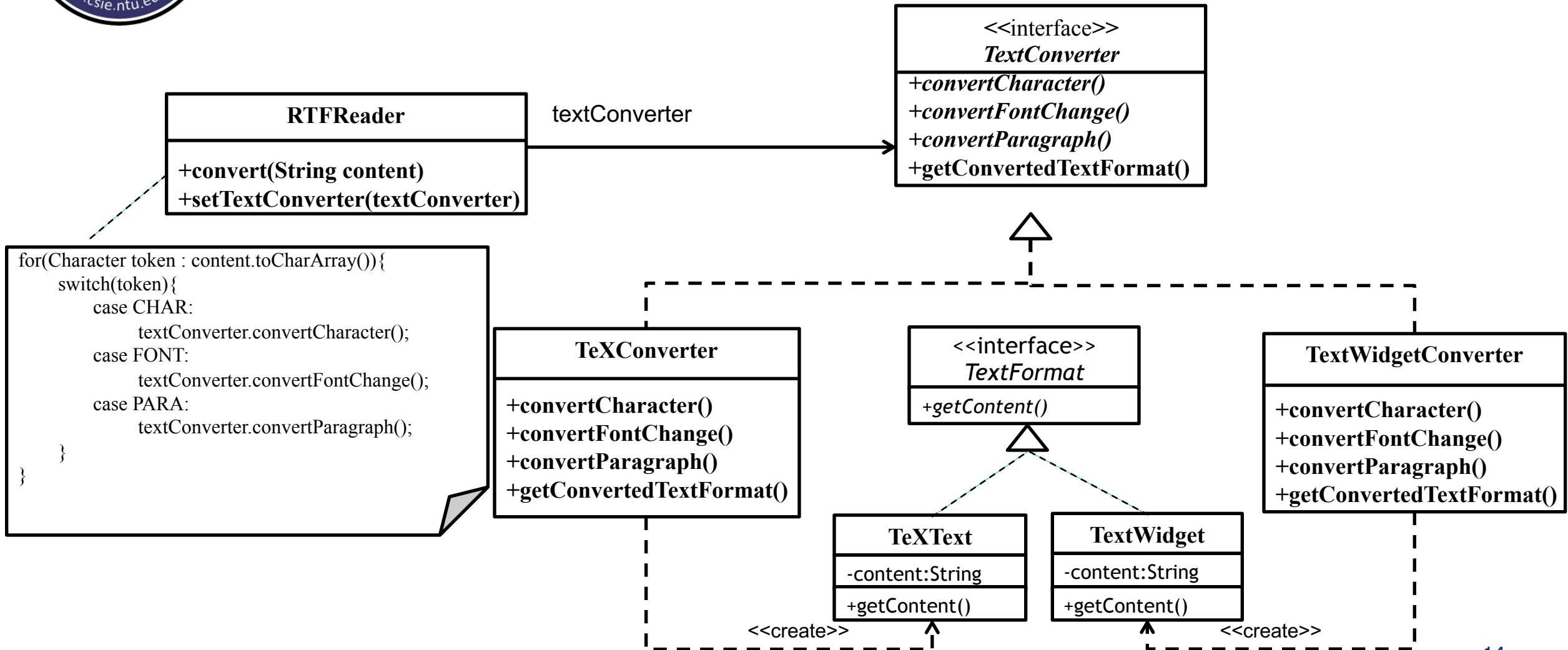
Act-3.3: Delegate behavior to an interface or an abstract class  
textConverter



```
for(Character token : content.toCharArray()){
    switch(token){
        case CHAR:
            textConverter.convertCharacter();
        case FONT:
            textConverter.convertFontChange();
        case PARA:
            textConverter.convertParagraph();
    }
}
```



# Refactored Design after Design Process





# Source code

## RTFReader

```
public class RTFReader {
    private TextConverter textConverter;

    public TextFormat convert(String content){
        for(Character token : content.toCharArray()){
            switch (token) {
                case 'C':
                    textConverter.convertCharacter();
                    break;
                case 'F':
                    textConverter.convertFontChange();
                    break;
                case 'P':
                    textConverter.convertParagraph();
                    break;
                default:
                    break;
            }
        }
        return textConverter.getConvertedTextFormat();
    }

    public void setTextConverter(TextConverter textConverter){
        this.textConverter = textConverter;
    }
}
```

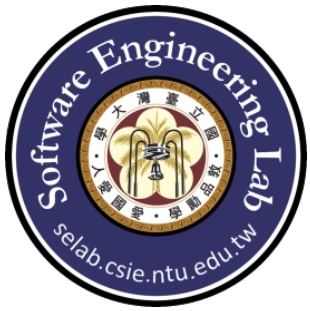




# Source code

## TextConverter

```
public interface TextConverter {  
    public void convertCharacter();  
    public void convertFontChange();  
    public void convertParagraph();  
    public TextFormat getConvertedTextFormat();  
}
```



# Source code

## TeXConverter

```
public class TeXConverter implements TextConverter{
    private String content = "";

    @Override
    public void convertCharacter() {
        content = content + 'c';
    }

    @Override
    public void convertFontChange() { content = content + '_'; }

    @Override
    public void convertParagraph() { content = content + '|'; }

    @Override
    public TeXText getConvertedTextFormat(){
        TeXText teXText = new TeXText(content);
        content = "";
        return teXText;
    }
}
```



# Source code

## TextWidgetConverter

```
public class TextWidgetConverter implements TextConverter{
    private String content = "";

    @Override
    public void convertCharacter() { content = content + "<Char>"; }

    @Override
    public void convertFontChange() { content = content + "<Font>"; }

    @Override
    public void convertParagraph() { content = content + "<Paragraph>"; }

    @Override
    public TextWidget getConvertedTextFormat(){
        TextWidget textWidget = new TextWidget(content);
        content = "";
        return textWidget;
    }
}
```



# Source code

## TextFormat

```
public interface TextFormat {  
    public String getContent();  
}
```

## TextWidget

```
public class TextWidget implements TextFormat{  
    private String content;  
    public TextWidget(String content) { this.content = content; }  
  
    public String getContent() { return content; }  
}
```

## TeXText

```
public class TeXText implements TextFormat{  
    private String content;  
    public TeXText(String content){  
        this.content = content;  
    }  
  
    public String getContent() { return content; }  
}
```



## Input:

[Text\_format]

[TRF\_token]

...

## Output:

[converted\_token]...

...



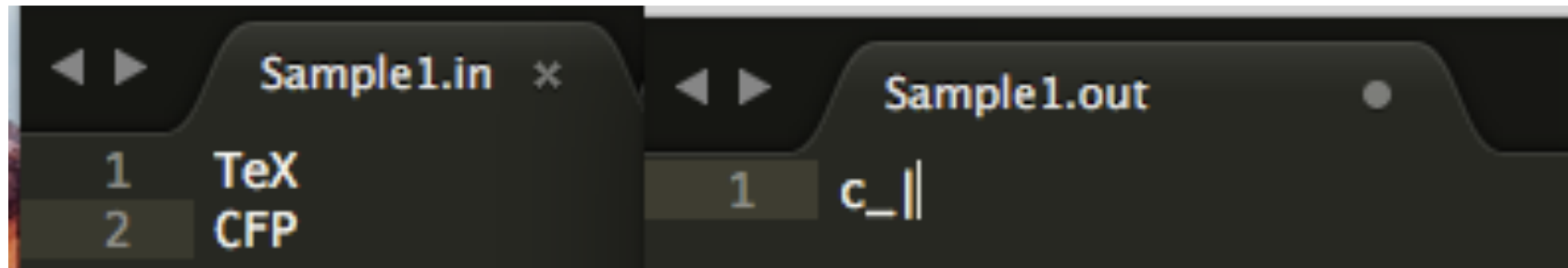
# Test cases

---

- ☐ TestCase 1: Three convert type for TeX
- ☐ TestCase 2: Three convert type for TextWidget
- ☐ TestCase 3: Complex



# Test case1







# Test case2

```
Sample2.in ×
1 TextWidget
2 CFP

Sample2.out
1 <Char><Font><Paragraph>
```



24



# Recurrent Problem

---

- ❑ Construct complex objects step-by-step, and return the product as a final step.



# Builder Pattern

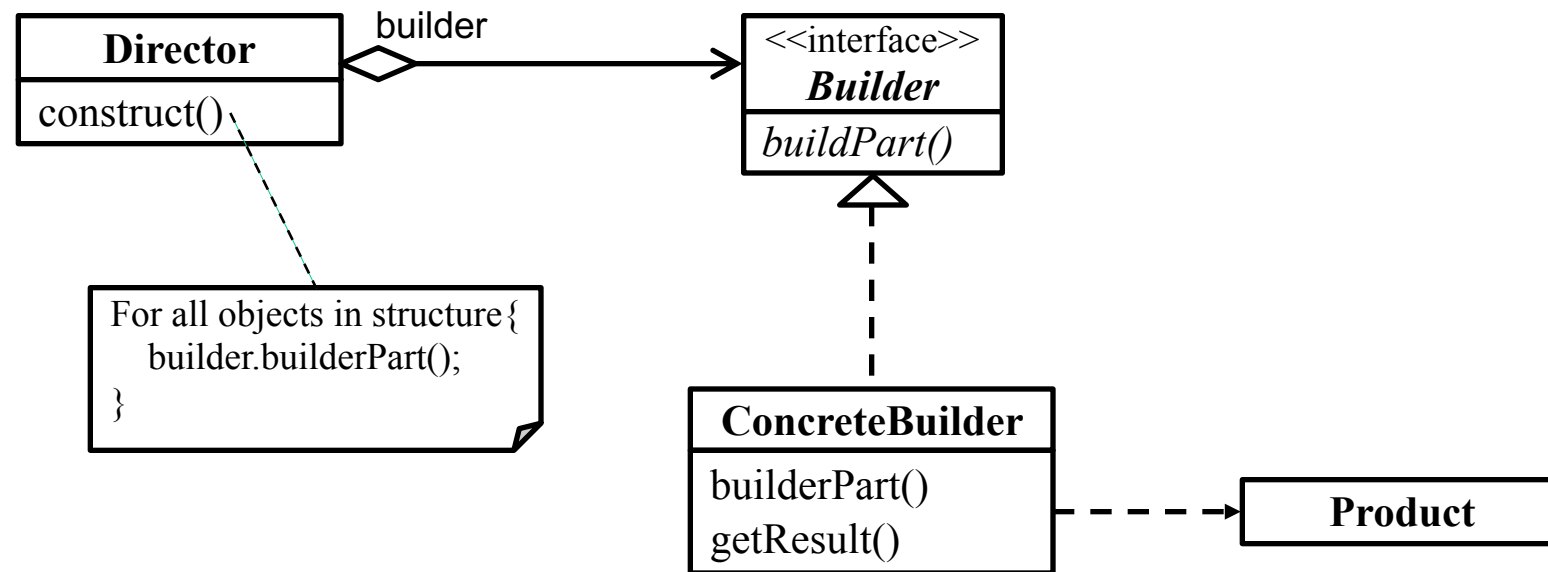
---

## □ Intent

- Separate the construction of a complex object from its representation so that the same construction process can create different representations.

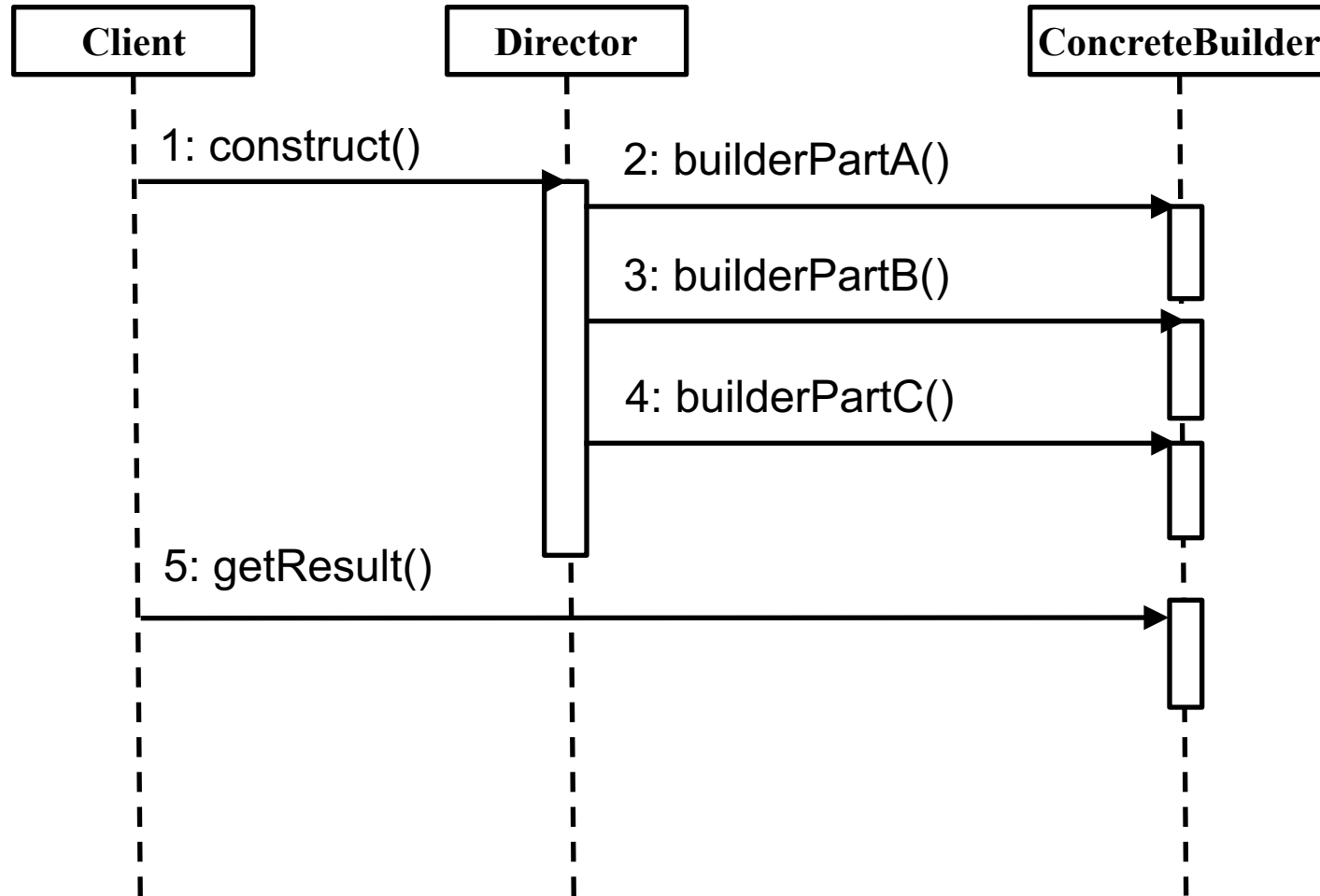


# Builder Pattern Structure<sub>1</sub>





# Builder Pattern Structure<sub>2</sub>





# Builder Pattern Structure<sub>3</sub>

	Instantiation	Use	Termination
Director	Don't Care	Don't Care	Don't Care
Builder	X	Director uses Builder to invoke ConcreteBuilders' method to build product through polymorphism.	X
ConcreteBuilder	Don't Care	Director invokes ConcreteBuilder's method to build product through polymorphism.	Don't Care
Product	ConcreteBuilder	Don't Care	Don't Care





# Abstract Factory vs. Builder<sub>1</sub>

## □ Abstract Factory

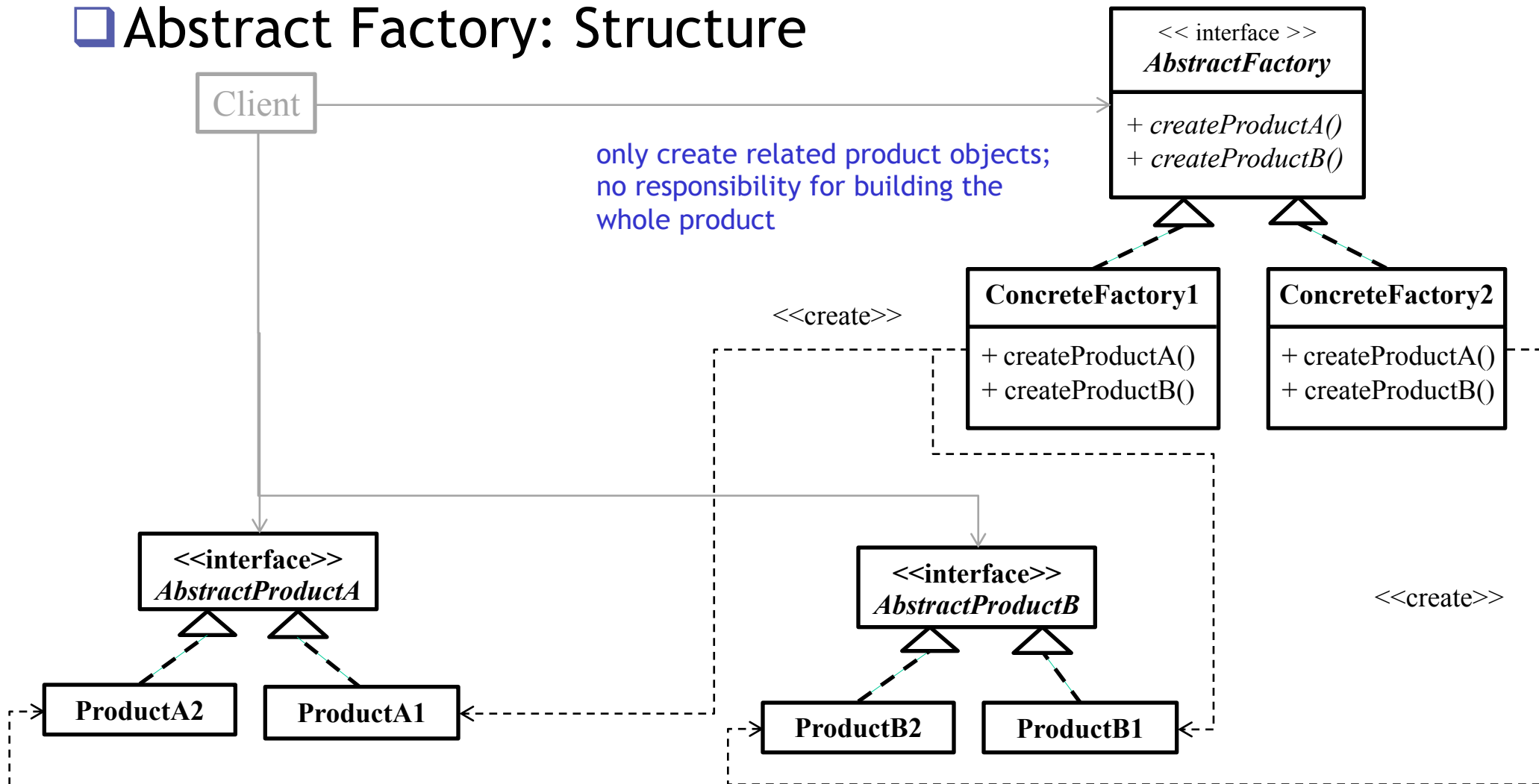
- families of product objects
- Create a set of related or dependent objects, and return each object once it is created

## □ Builder

- how a composite object gets created
- Create a set of related or dependent objects, and provide an interface to return the final product

# Abstract Factory vs. Builder<sub>2</sub>

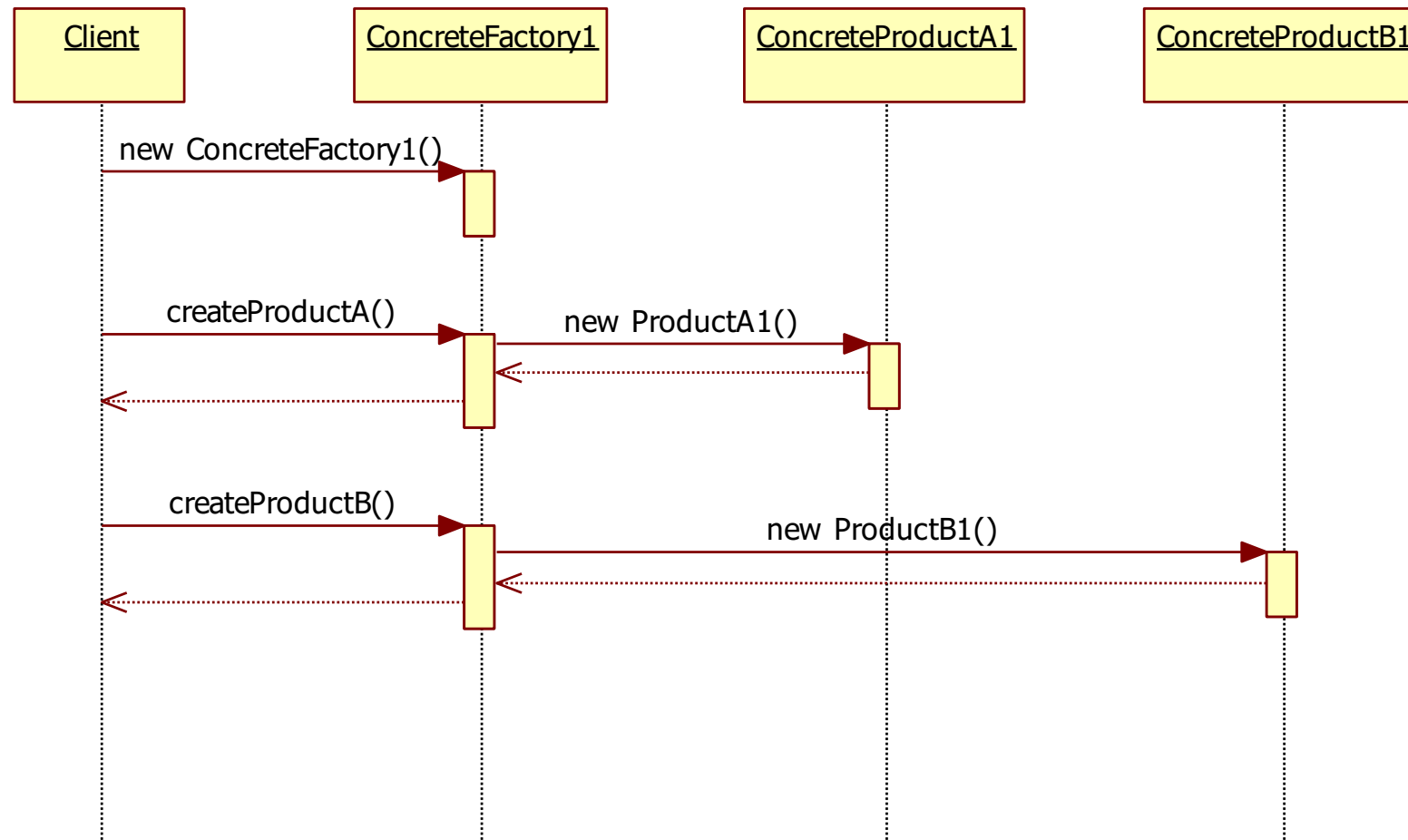
## □ Abstract Factory: Structure





# Abstract Factory vs. Builder<sub>3</sub>

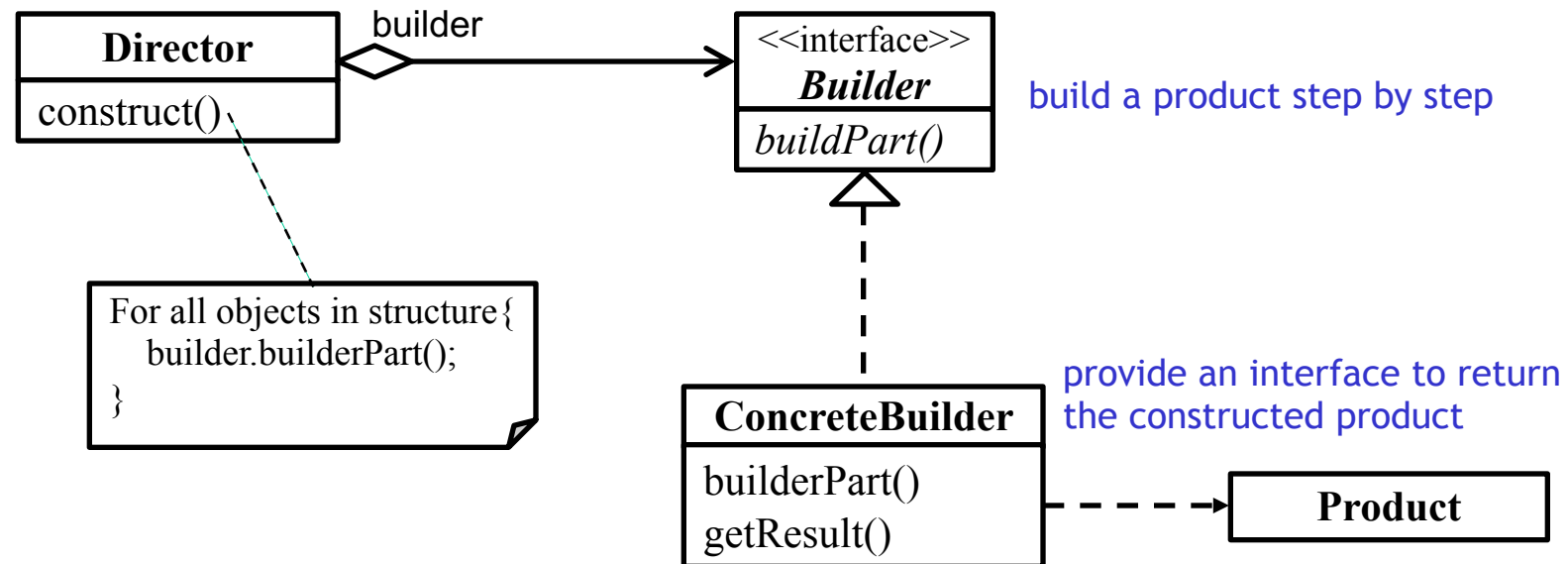
## □ Abstract Factory: Sequence Diagram





# Abstract Factory vs. Builder<sub>4</sub>

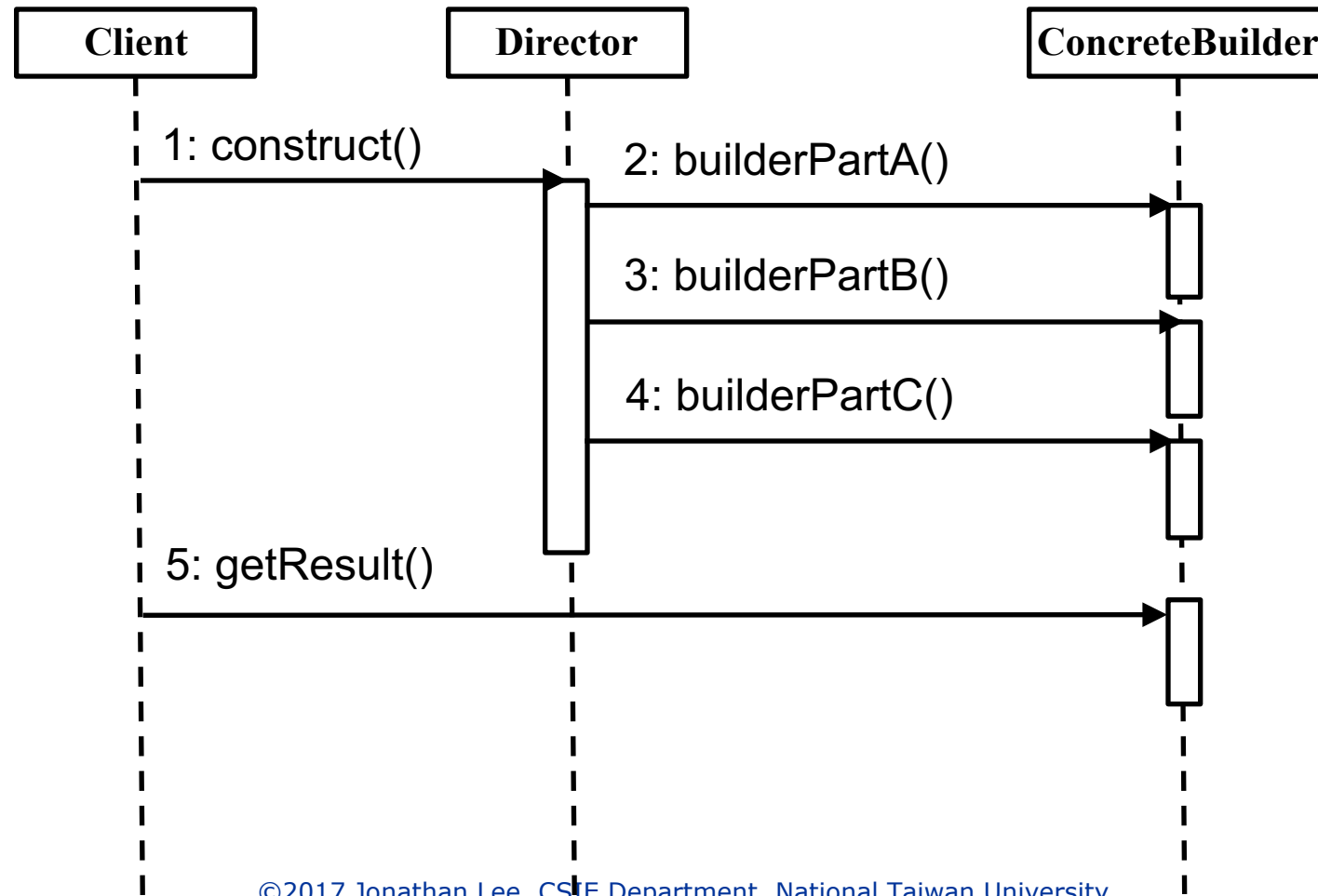
## □ Builder: Structure

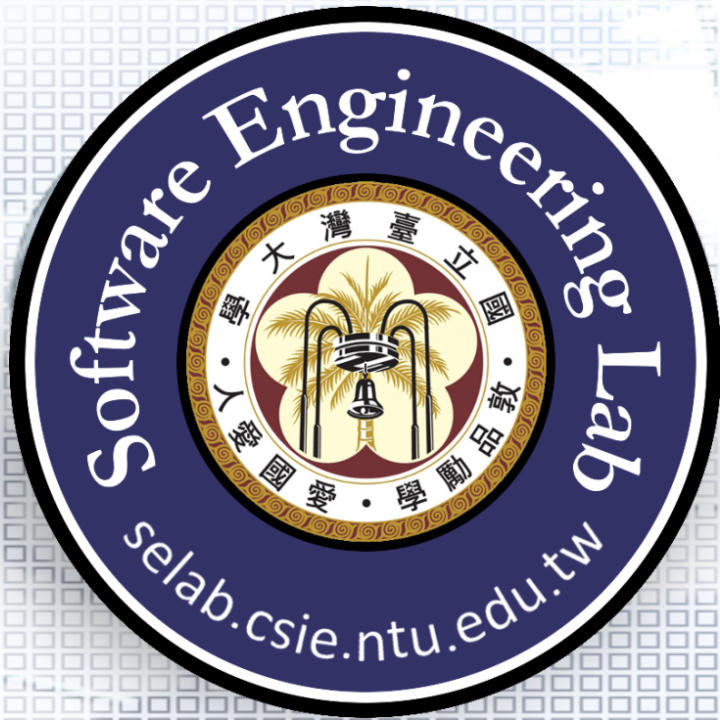




# Abstract Factory vs. Builder<sub>5</sub>

## ❑ Builder: Sequence Diagram





# A Vacation Planner (Builder)

Prof. Jonathan Lee (李允中)

Department of Computer Science and  
Information Engineering  
National Taiwan University



# Requirements Statement

---

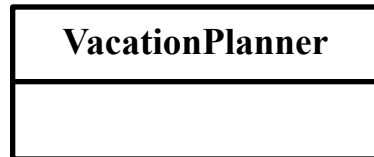
- ☐ Patternsland wants to build a vacation planner.
- ☐ A vacation planner can choose a hotel and various types of admission tickets, make restaurant reservations, even book special events and day. The different types of vacations, such as normal and backpacking, can have different options.





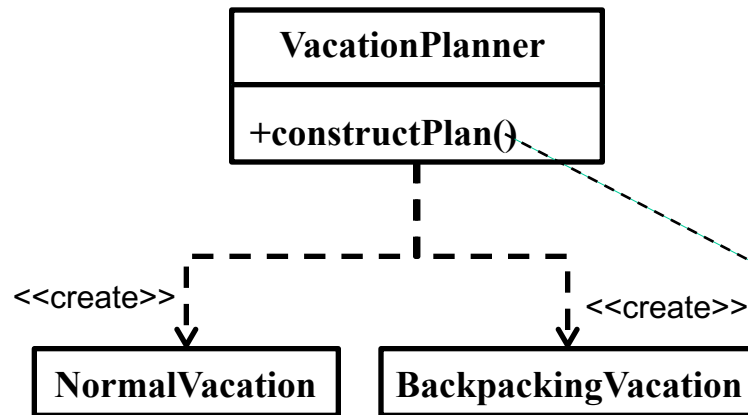
# Requirements Statements<sub>1</sub>

- ❑ Patternsland wants to build a vacation planner.



# Requirements Statements<sub>2</sub>

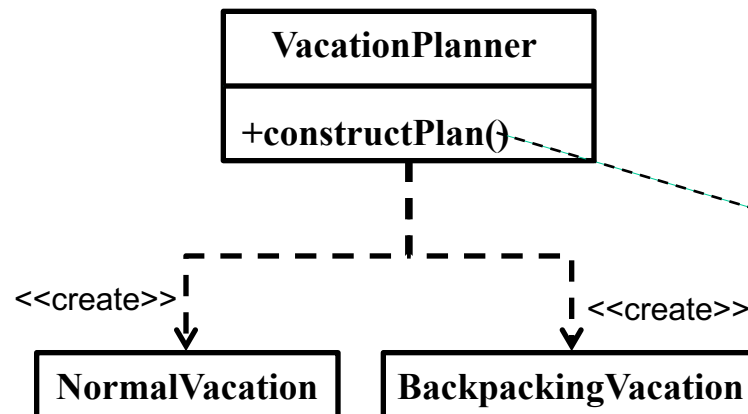
- A vacation planner can choose a hotel and various types of admission tickets, make restaurant reservations, even book special events and day. The difference types vacation (normal, backpacking) will have different options.



```
if(type=="normal"){
//choose Day
//choose Hotel
//choose Reservation
//choose Special Event
//choose Tickets
//create and return Normal Vacation Plan
}else if(type=="backpacking"){
//choose Day
//choose Hotel
//choose Reservation
//choose Special Event
//choose Tickets
//create and return Backpacking Vacation Plan
}
```

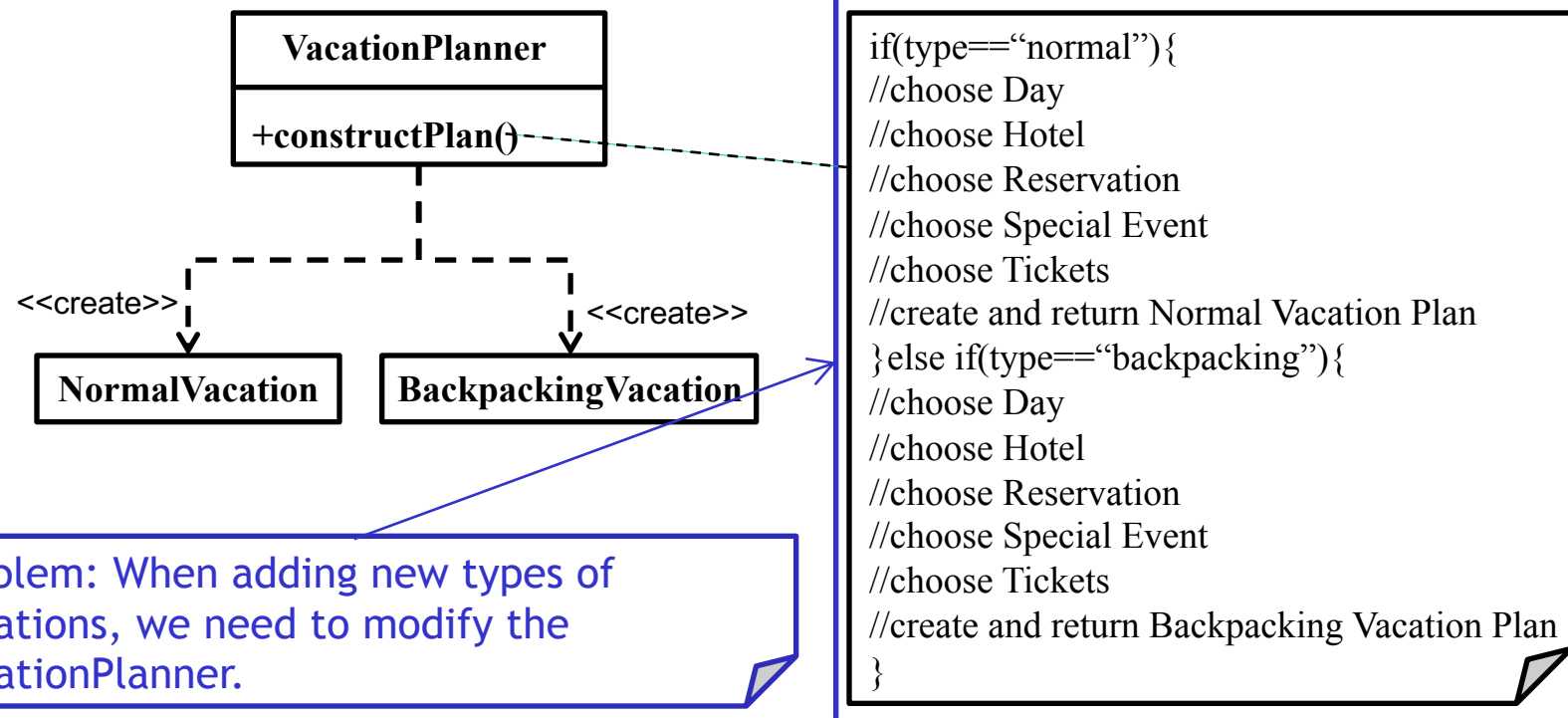


# Initial Design - Class Diagram

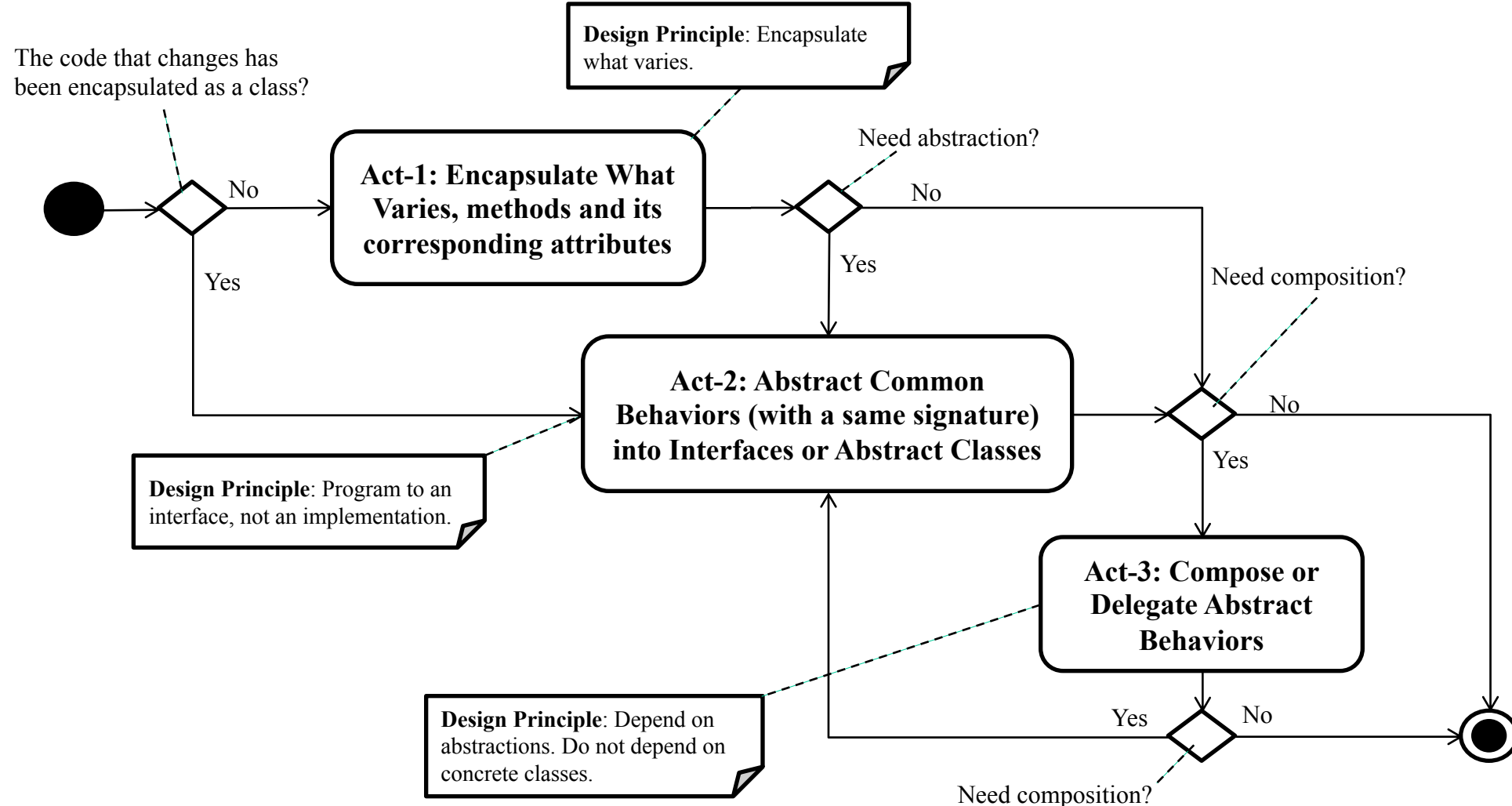


```
if(type=="normal"){
//choose Day
//choose Hotel
//choose Reservation
//choose Special Event
//choose Tickets
//create and return Normal Vacation Plan
}else if(type=="backpacking"){
//choose Day
//choose Hotel
//choose Reservation
//choose Special Event
//choose Tickets
//create and return Backpacking Vacation Plan
}
```

# Problems with Initial Design

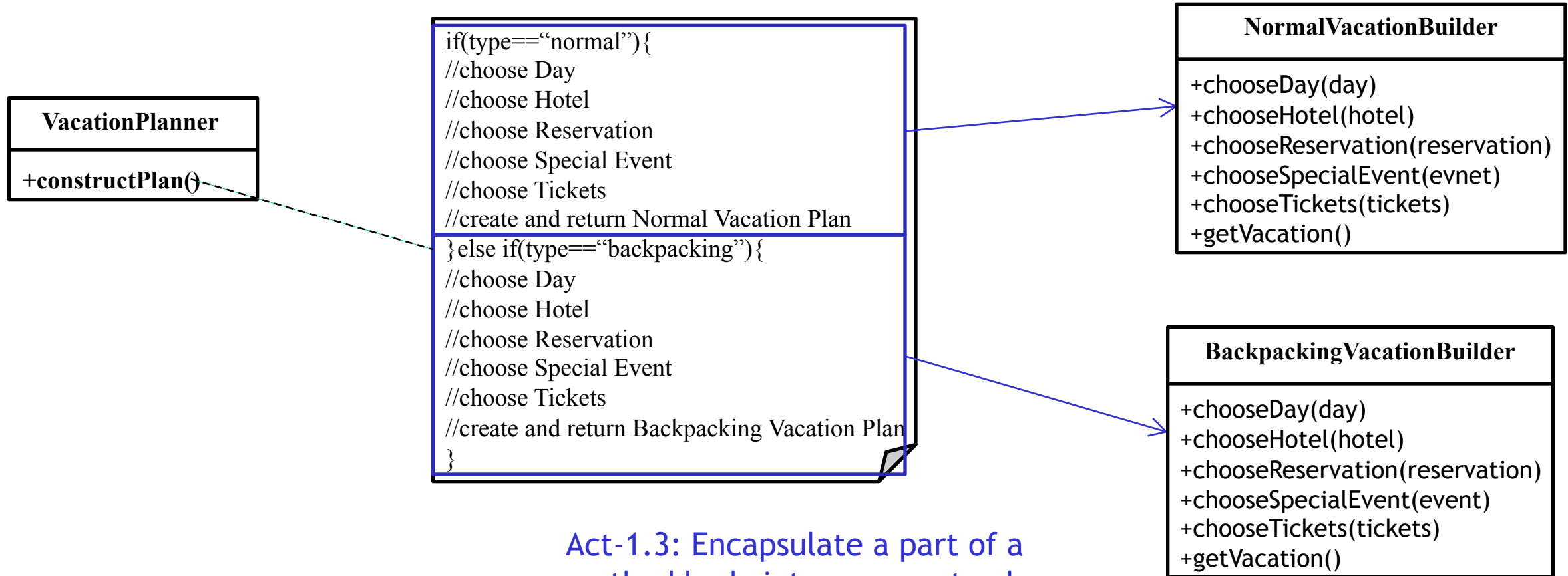


# Design Process for Change





# Act-1: Encapsulate What Varies

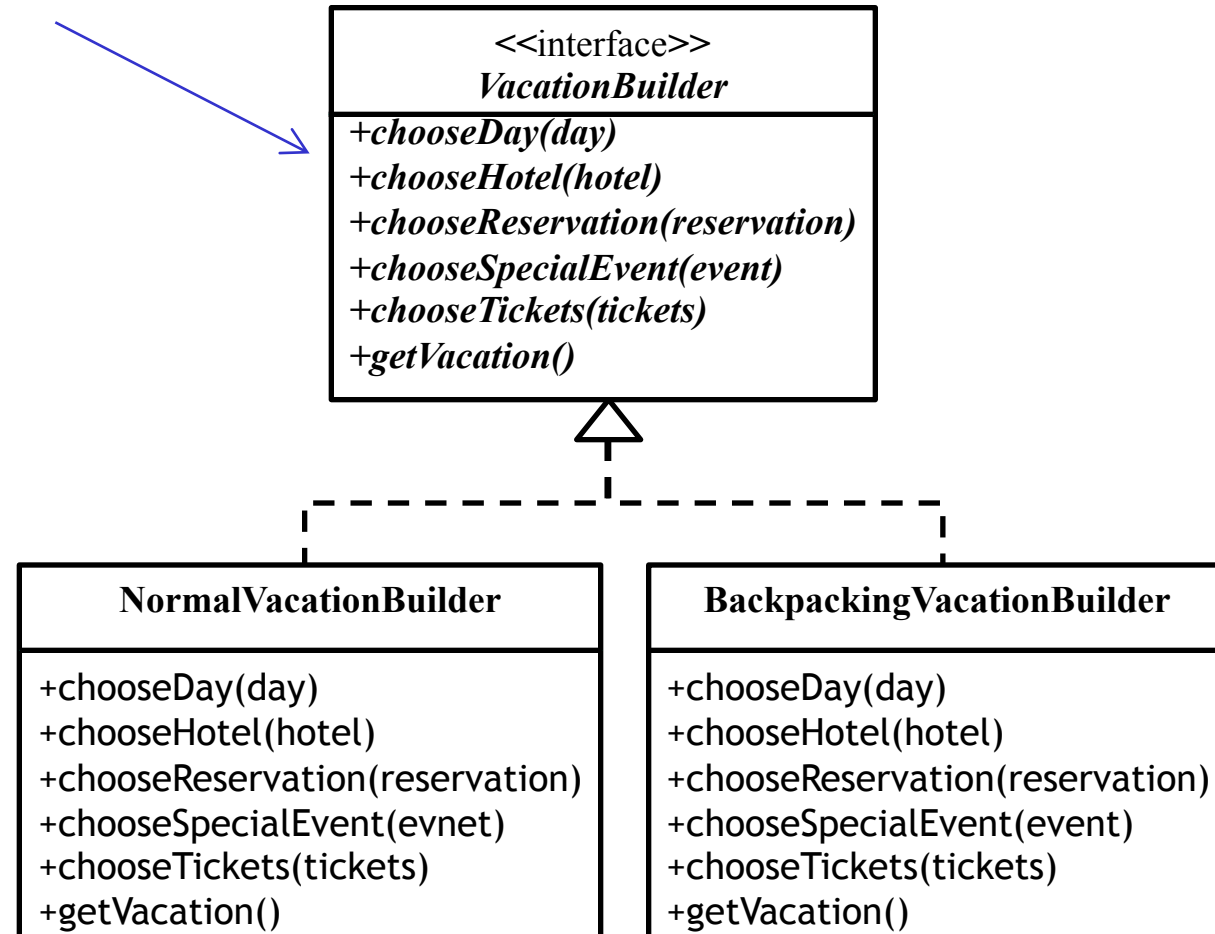


Act-1.3: Encapsulate a part of a method body into a concrete class



# Act-2: Abstract Common Behaviors

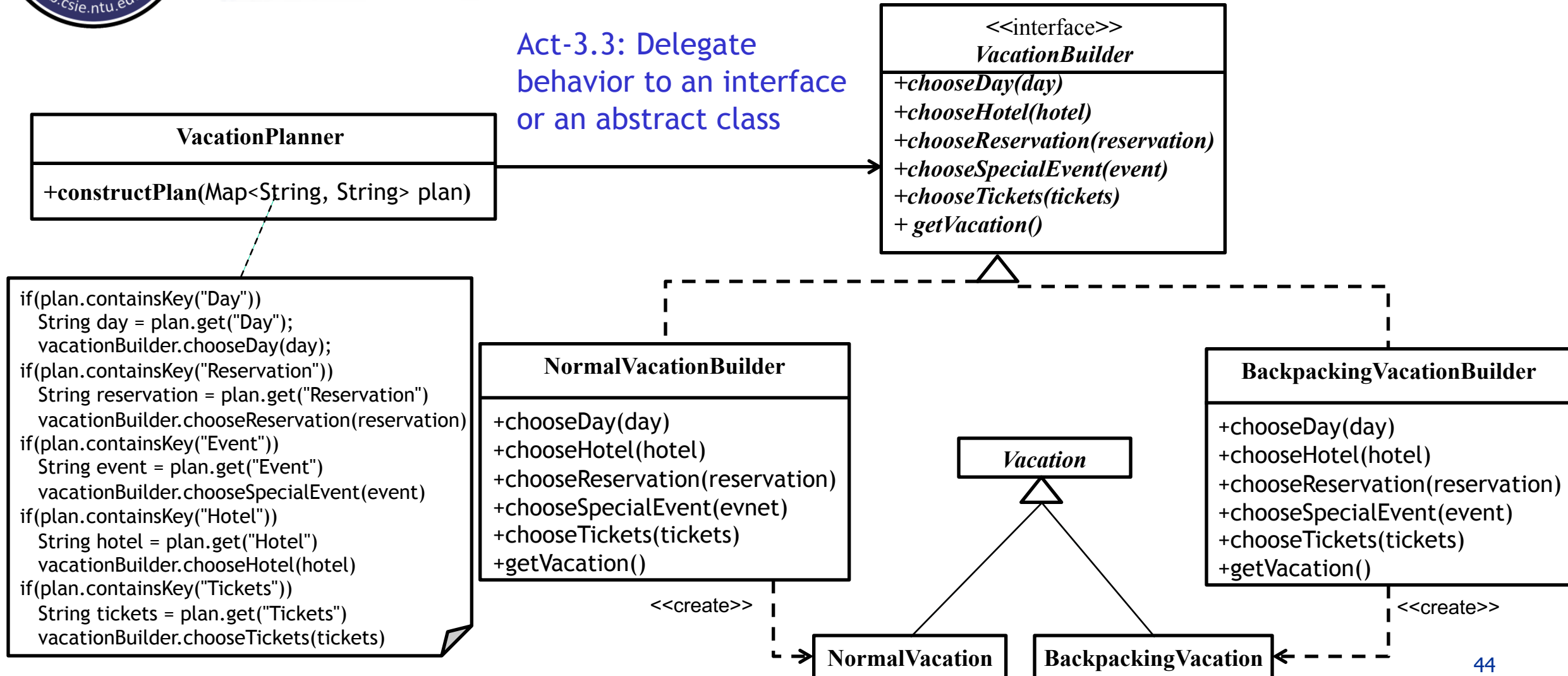
Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism





# Act-3: Compose Abstract Behaviors

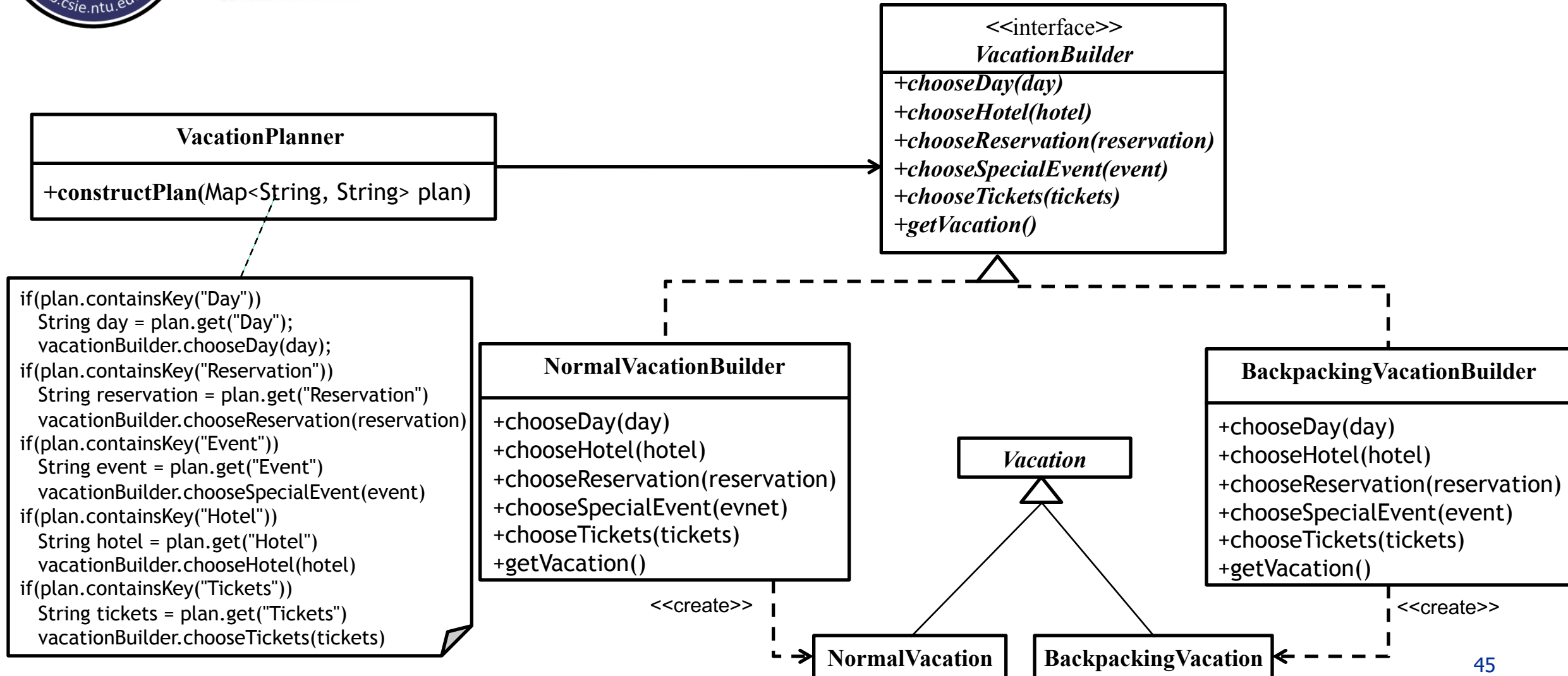
## Act-3.3: Delegate behavior to an interface or an abstract class







# Refactored Design after Design Process





# VacationPlanner

```
public class VacationPlanner {
    private VacationBuilder vacationBuilder = null;

    public void setVacationBuilder(VacationBuilder vacationBuilder) { this.vacationBuilder = vacationBuilder; }

    public Vacation constructPlan(Map<String, String> plan){
        if(plan.containsKey("Day")){
            String day = plan.get("Day");
            vacationBuilder.chooseDay(day);
        }
        if(plan.containsKey("Reservation")){
            String reservation = plan.get("Reservation");
            vacationBuilder.chooseReservation(reservation);
        }
        if(plan.containsKey("Event")){
            String event = plan.get("Event");
            vacationBuilder.chooseSpecialEvent(event);
        }
        if(plan.containsKey("Hotel")){
            String hotel = plan.get("Hotel");
            vacationBuilder.chooseHotel(hotel);
        }
        if(plan.containsKey("Tickets")){
            String tickets = plan.get("Tickets");
            vacationBuilder.chooseTickets(tickets);
        }
        return vacationBuilder.getVocation();
    }
}
```



# VacationBuilder

```
public interface VacationBuilder {  
    public void chooseDay(String day);  
    public void chooseHotel(String hotel);  
    public void chooseReservation(String reservation);  
    public void chooseSpecialEvent(String event);  
    public void chooseTickets(String tickets);  
    public Vacation getVocation();  
}
```



# NormalVacationBuilder

```
public class NormalVacationBuilder implements VacationBuilder{
    private String day = "";
    private String hotel = "";
    private String reservation = "";
    private String event = "";
    private String tickets = "";

    @Override
    public Vacation getVocation(){ return new NormalVacation(day, hotel, reservation, event, tickets);}

    @Override
    public void chooseDay(String day){ this.day = day;}

    @Override
    public void chooseHotel(String hotel){ this.hotel = hotel;}

    @Override
    public void chooseReservation(String reservation){ this.reservation = reservation;}

    @Override
    public void chooseSpecialEvent(String event){ this.event = event;}

    @Override
    public void chooseTickets(String tickets){ this.tickets = tickets;}
}
```



# BackpackingVacationBuilder

```
public class BackpackingVacationBuilder implements VacationBuilder{
    private String day = "";
    private String hotel = "";
    private String reservation = "";
    private String event = "";
    private String tickets = "";

    @Override
    public Vacation getVocation(){ return new BackpackingVacation(day, hotel, reservation, event, tickets); }

    @Override
    public void chooseDay(String day){ this.day = day; }

    @Override
    public void chooseHotel(String hotel){ this.hotel = hotel; }

    @Override
    public void chooseReservation(String reservation){ this.reservation = reservation; }

    @Override
    public void chooseSpecialEvent(String event){ this.event = event; }

    @Override
    public void chooseTickets(String tickets){ this.tickets = tickets; }
}
```



# Vacation

```
public abstract class Vacation {
    private String day = "";
    private String hotel = "";
    private String reservation = "";
    private String specialEvent = "";
    private String tickets = "";

    public Vacation(String day, String hotel, String reservation, String specialEvent, String tickets) {
        this.day = day;
        this.hotel = hotel;
        this.reservation = reservation;
        this.specialEvent = specialEvent;
        this.tickets = tickets;
    }

    @Override
    public String toString(){
        String string = getClass().getName() + "\n" +
            "Day:" + day + "\n" +
            "Hotel:" + hotel + "\n" +
            "Reservation:" + reservation + "\n" +
            "SpecialEvent:" + specialEvent + "\n" +
            "Tickets:" + tickets + "\n";

        return string;
    }
}
```



# Source code

## BackpackingVacation

```
public class BackpackingVacation extends Vacation{  
    public BackpackingVacation(String day, String hotel, String reservation, String specialEvent, String tickets)  
        super(day, hotel, reservation, specialEvent, tickets);  
}  
}
```

## NormalVacation

```
public class NormalVacation extends Vacation{  
    public NormalVacation(String day, String hotel, String reservation, String specialEvent, String tickets) {  
        super(day, hotel, reservation, specialEvent, tickets);  
    }  
}
```



# Input / Output

## Input:

```
[Vacation_type]

Day [day]

SpecialEvent [event]

Hotel [hotel]

Reservation [reservation]

Tickets [tickets]

// input information order could be different from above.

...
```

## Output:

```
[Vacation_type]Vacation

Day:[day]

Hotel:[hotel]

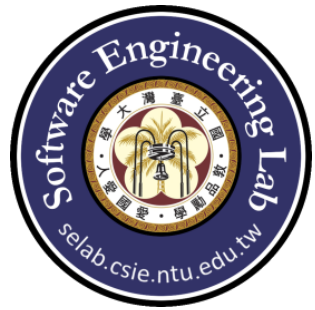
Reservation:[reservation]

SpecialEvent:[event]

Tickets:[tickets]

//output order must be the same as above.
```





# Test cases

---

- ☐ TestCase 1:      only Normal
- ☐ TestCase 2:      only Backpacking
- ☐ TestCase 3:      Both Normal and Backpacking
- ☐ TestCase 4:      Complex



# Test case1

Sample1.in *		Sample2.	Sample1.out *	
1	Normal		1	NormalVacation
2	Day Monday		2	Day:Monday
3	SpecialEvent eventA		3	Hotel:hotelA
4	Hotel hotelA		4	Reservation:reservationA
5	Reservation reservationA		5	SpecialEvent:eventA
6	Tickets TicketsA		6	Tickets:TicketsA



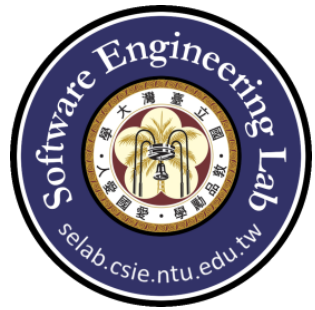
# Test case2

Sample2.in	Sample3.in	Sample2.out
1 Backpacking		1 BackpackingVacation
2 Day Monday		2 Day:Monday
3 SpecialEvent eventA		3 Hotel:hotelA
4 Hotel hotelA		4 Reservation:reservationA
5 Reservation reservationA		5 SpecialEvent:eventA
6 Tickets TicketsA		6 Tickets:TicketsA



# Test case3

Sample3.in	Sample3.out
1 Normal	1 NormalVacation
2 Day Monday	2 Day:Monday
3 SpecialEvent eventA	3 Hotel:hotelA
4 Hotel hotelA	4 Reservation:reservationA
5 Reservation reservationA	5 SpecialEvent:eventA
6 Tickets TicketsA	6 Tickets:TicketsA
7 Backpacking	7 BackpackingVacation
8 Day TuesDay	8 Day:TuesDay
9 SpecialEvent eventB	9 Hotel:hotelB
10 Hotel hotelB	10 Reservation:reservationB
11 Reservation reservationB	11 SpecialEvent:eventB
12 Tickets TicketsB	12 Tickets:TicketsB
13 Normal	13 NormalVacation
14 Day Sunday	14 Day:Sunday
15 SpecialEvent eventC	15 Hotel:hotelC
16 Hotel hotelC	16 Reservation:reservationC
17 Reservation reservationC	17 SpecialEvent:eventC
18 Tickets TicketsC	18 Tickets:TicketsC



# Test case4

Sample4.in	Sample4.out
1 Normal	1 NormalVacation
2 Day Monday	2 Day:Monday
3 SpecialEvent eventA	3 Hotel:hotelA
4 Hotel hotelA	4 Reservation:reservationA
5 Reservation reservationA	5 SpecialEvent:eventA
6 Tickets TicketsA	6 Tickets:TicketsA
7 Backpacking	7 BackpackingVacation
8 SpecialEvent eventB	8 Day:
9 Hotel hotelB	9 Hotel:hotelB
10 Tickets TicketsB	10 Reservation:reservationB
11 Reservation reservationB	11 SpecialEvent:eventB
12 Normal	12 Tickets:TicketsB
13 Day Sunday	13 NormalVacation
14 Hotel hotelC	14 Day:Sunday
15 Tickets TicketsC	15 Hotel:hotelC
16 SpecialEvent eventC	16 Reservation:
	17 SpecialEvent:eventC
	18 Tickets:TicketsC