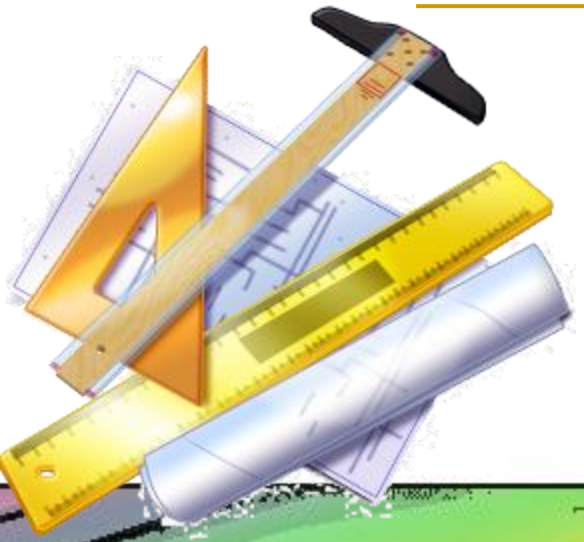
ทบทวนการออกแบบวัตถุและตัวอย่าง



Lecture 13

เยาวดี เต็มธนาภัทร์

Yaowadee Temtanapat

จุดมุ่งหมายของบทนี้

- ทบทวนหลักการออกแบบเชิงวัตถุ
- ทบทวนการระบุหาวัตถุจากปัญหา
- แสดงตัวอย่างการออกแบบเชิงวัตถุ

หลักการเชิงวัตถุ

- **การกำหนดสาระสำคัญ (*Abstraction*)**
  - การกาหนดสิ่งต่าง ๆ โดยเน้นไปที่สาระสำคัญและละเว้นรายละเอียดปลีกย่อย
- **การประกอบ (*Composition*)**
  - การสร้างซอฟต์แวร์โดยการใช้องค์ประกอบย่อยรวมกันเข้าเพื่อแก้ปัญหาที่ต้องการ
- **การห่อหุ้มและการซ่อนข้อมูล (*Encapsulation and Information Hiding*)**
  - การแยกส่วนของวัตถุในแง่มุมมองภายนอกกับมุมมองภายใน เพื่อห่อหุ้มและซ่อนสิ่งที่ไม่จำเป็นต้องรู้จากมุมมองภายนอก
- **ลำดับชั้น (*Hierarchy*)**
  - การจัดลำดับชั้นของวัตถุเพื่อให้ง่ายในการเข้าใจและขยายต่อได้

วัตถุ

- การโปรแกรมเชิงวัตถุ คือการสร้างโมเดลจากวัตถุ
- วัตถุ
  - อาจเป็นสิ่งจับต้องได้ เช่น สินค้า
  - อาจเป็นนามธรรม เช่น การนดั หมาย
  - อาจเป็นกระบวนการทางาน เช่น การประมวลผลเกรด
- องค์ประกอบของวัตถุ
  - พฤติกรรม หรือ Method: กิจกรรมที่วัตถุนั้นรู้หรือว่าสามารถทำได้
  - ลักษณะ หรือ Attribute: ส่วนที่ประกอบเป็นวัตถุ ค่าอาจเปลี่ยนไปตามเวลาและวัตถุ

การทดสอบความเป็นวัตถุ

- ความเกี่ยวข้องกับปัญหา
  - อยู่ในขอบเขตของปัญหา
  - รับผิดชอบในการทำหน้าที่อย่างหนึ่งอย่างใดในปัญหา
  - อาจเป็นส่วนหนึ่งของวัตถุอื่นที่จำเป็นในปัญหา
- มีความเป็นเอกเทศจากวัตถุอื่น
  - ปรากฏอยู่ได้ด้วยตัวเอง และมีความเป็น modularity
- มี attributes และ methods ของตัวเอง

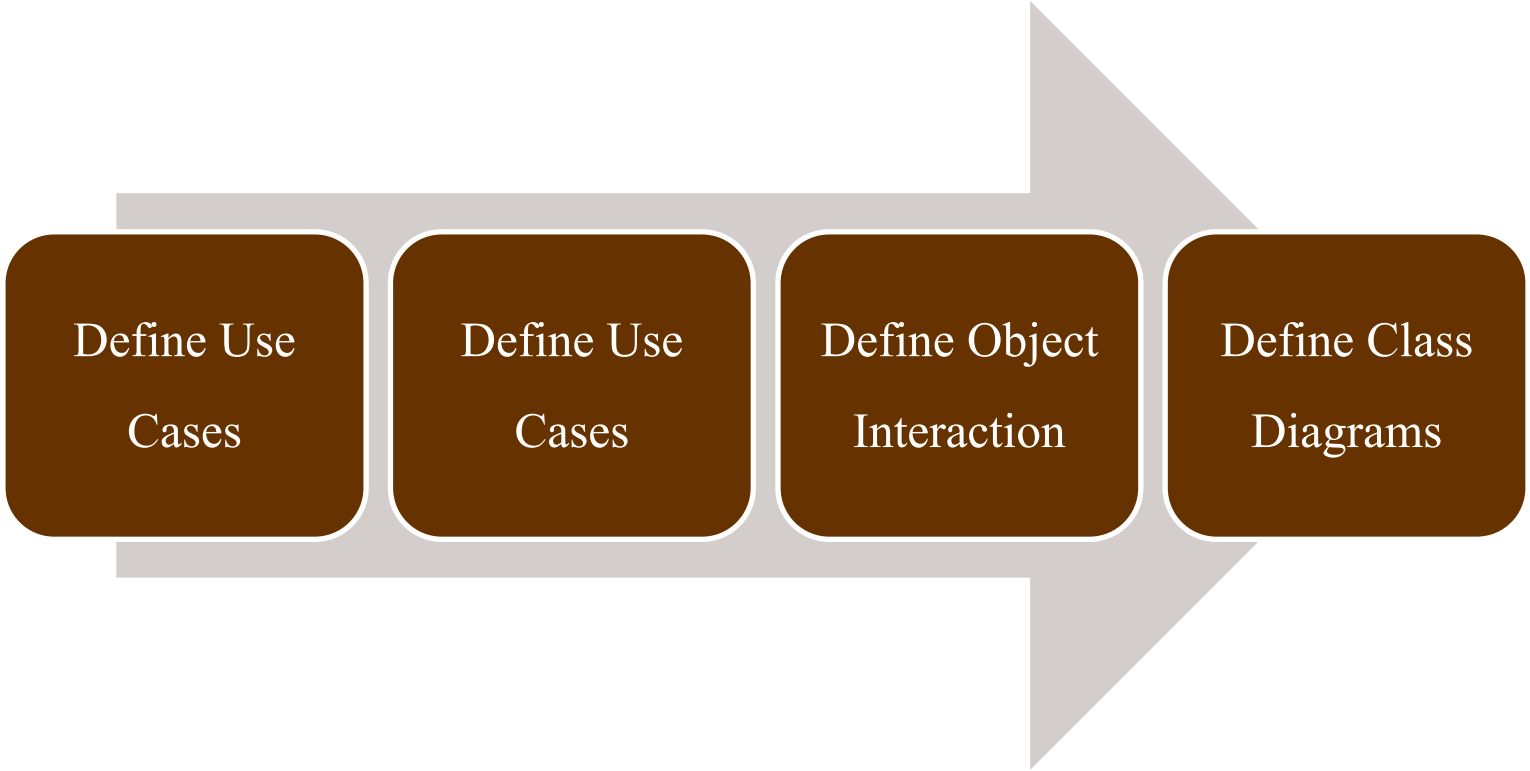
วัตถุควรมีความเป็น Modularity

- ลักษณะของการแยกวัตถุภายในระบบที่ดี
  - วัตถุควรมีลักษณะที่เบ็ดเสร็จภายในตัวเองและง่ายในการบริหารจัดการ
  - cohesiveness สูงและมี coupling แบบหลวม ๆ
- *Cohesiveness*: วัดความสัมพันธ์ภายในวัตถุเดียวกัน (intra-object relatedness)
  - ทั่วไปแสดงในรูป Interface จึงควรเป็น concept เดียวที่เกี่ยวกับวัตถุนั้นเท่านั้น
- *Coupling*: วัดความเป็นอิสระระหว่างวัตถุที่ต่างกัน
  - ทั่วไปแสดงถึงการขึ้นต่อกันของวัตถุกับวัตถุอื่น

การวิเคราะห์และการออกแบบ (Analysis & Design)

- กฎพื้นฐาน: พยายามค้นหา
  - **ค้นหา class:** ว่าอะไรคือ objects, ต้องแบ่ง project ของเราอย่างไรให้เป็นส่วนย่อย (component parts)
  - **กำหนดพฤติกรรมของแต่ละ class:** ต้องมี method อะไรบ้างใน class
  - **กำหนดความสัมพันธ์ระหว่าง class:** อะไรบ้างที่เป็นส่วน interfaces ของ object เหล่านี้, messages อะไรบ้างที่ต้องการเพื่อทำให้สามารถติดต่อกันได้ระหว่าง object

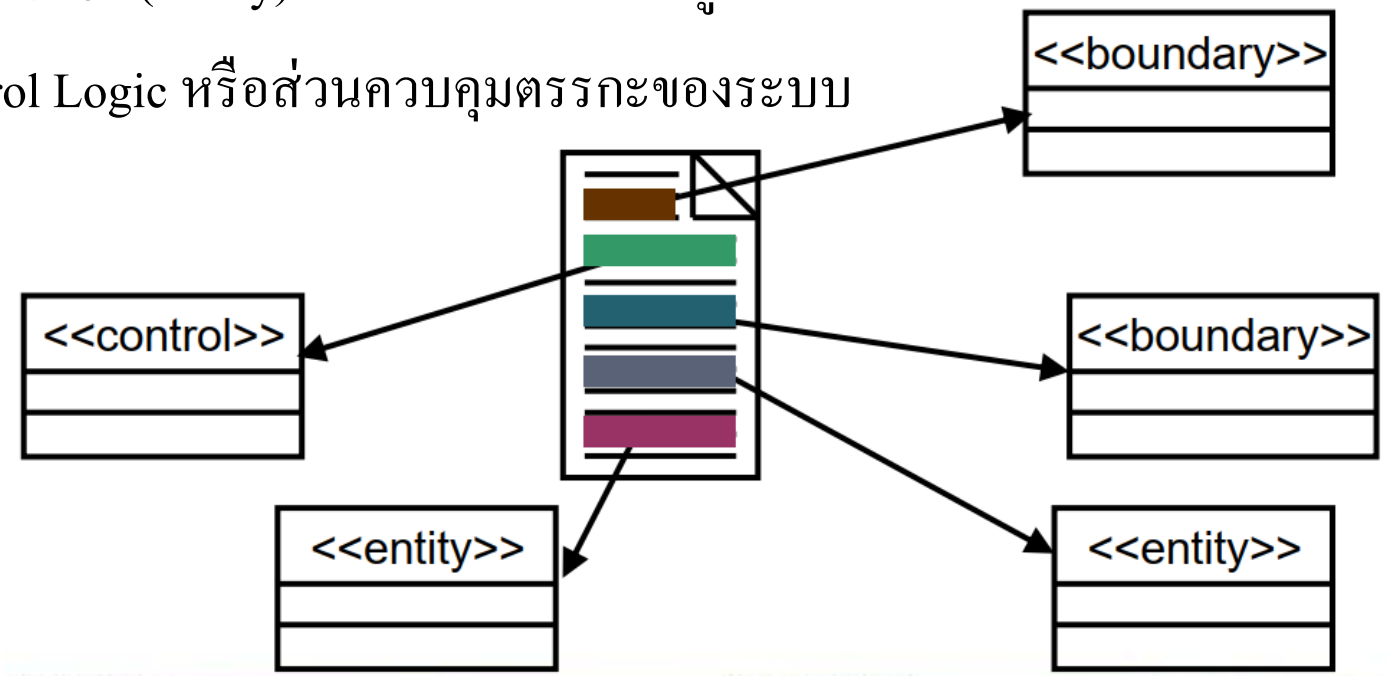
ขั้นตอนในการวิเคราะห์และออกแบบอย่างง่าย



- ในแต่ละขั้นตอนสามารถใช้แผนภาพ UML เพื่อช่วยในการอธิบาย
- ระบุคลาส และกระจายคุณสมบัติและการทำงานในแต่ละ use case ไปตามคลาสต่าง ๆ

การระบุคลาสจากพฤติกรรมระบบ

- สามารถแบ่งมุมมองของระบบเพื่อระบุคลาสที่เป็นไปได้
  - Boundary หรือคลาสที่เป็นขอบเขตระหว่างระบบกับผู้ใช้งาน อุปกรณ์ หรือระบบอื่น ๆ เช่น UI, Menu, I/O
  - Information (Entity) หรือตัวแทนของข้อมูลที่ระบบใช้
  - Control Logic หรือส่วนควบคุมตรรกะของระบบ



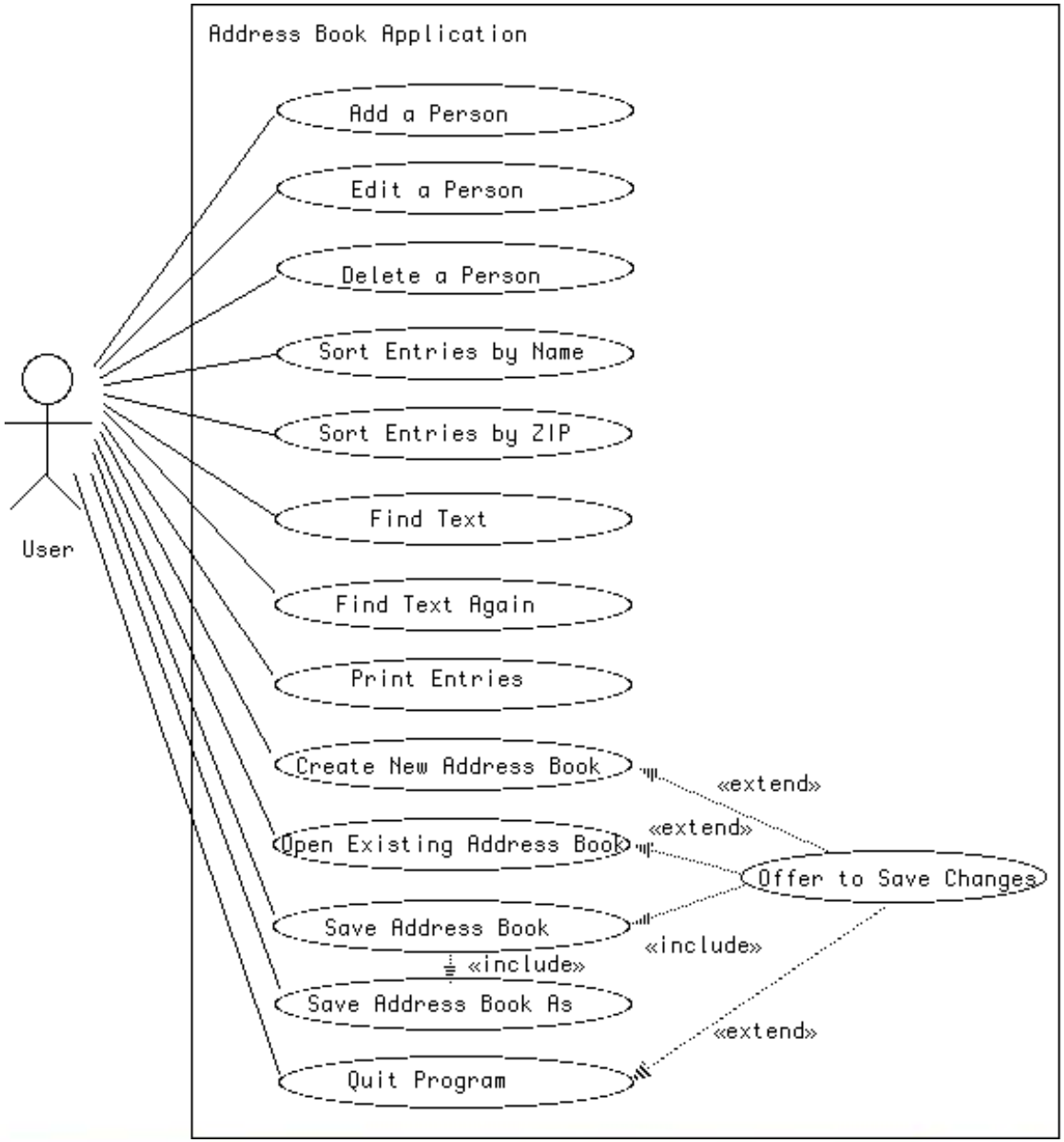
ตัวอย่างการออกแบบโปรแกรม Address Book

- จาก http://www.cs.gordon.edu/courses/cs211/AddressBookExample/

ความต้องการ

- โปรแกรมสมุดจดที่อยู่ (Address Book) ที่สามารถเก็บข้อมูลติดต่อรายบุคคล โดยแต่ละรายการจะประกอบด้วย ชื่อ นามสกลุ ที่อยู่ เมือง รหัสไปรษณีย์ และเบอร์โทรศัพท์
- โปรแกรมสามารถเพิ่มข้อมูลบุคคลใหม่ได้และแก้ไข (ยกเว้นชื่อ) และลบข้อมูลของบุคคลเดิมได้ นอกจากนั้นยังสามารถจัดเรียงข้อมูลบุคคลใน Address Book ตามลำดับตัวอักษร โดยใช้นามสกุลเป็นหลัก (ถ้านามสกลุ เหมือนกันให้ใช้ชื่อช่วยในการเรียง) หรือด้วยรหัสไปรษณีย์
- สามารถสร้าง Address Book ใหม่ หรือเปิด Address Book เดิมโดยการเปิดไฟล์ที่มีข้อมูล Address Book อยู่ และสามารถ บันทึกและปิด Address Book ที่เปิดลงไฟล์ได้ โดยมีเมนู เปิด ปิด บันทึก และบันทึกเป็น เพื่อใช้ทำางานในส่วนการจัดการ Address Book
- ในเวอร์ชันแรกโปรแกรมจะเปิดไฟล์ Address Book ได้ทีละไฟล์เท่านั้นและต้องปิดไฟล์เก่าก่อนเปิดไฟล์ใหม่ แต่มีแผนให้ในเวอร์ชันถัดไป โปรแกรมสามารถเปิด Address ได้พร้อม ๆ กันทีเดียวหลายไฟล์ในหน้าต่างของตัวเอง โปรแกรมจะปิดเมื่อหน้าต่างสุดท้ายโดนปิด
- โปรแกรมจะตรวจว่ามีการเปลี่ยนแปลงที่เกิดขึ้นหลังจากบันทึกครั้งสุดท้ายไหม ถ้ามี เมื่อ Address Book จะถูกปิด ไม่ว่าจะจากการเปิดไฟล์ใหม่ หรือปิดโปรแกรม จะแจ้งเตือนผู้ใช้ให้บันทึก

Use Case

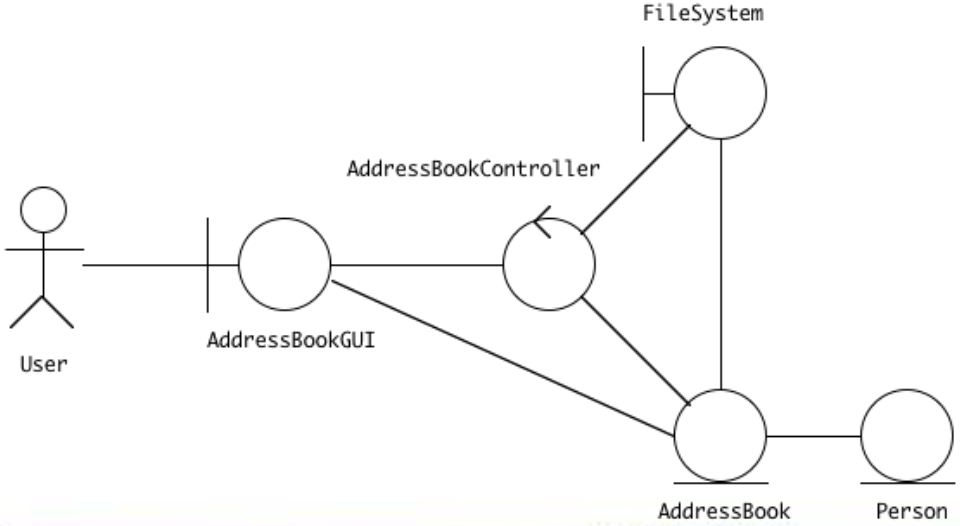

Address Book Application
Add a Person
Edit a Person
Delete a Person
Sort Entries by Name
Sort Entries by ZIP
Find Text
Find Text Again
Print Entries
Create New Address Book
Open Existing Address Book
Save Address Book
Save Address Book As
Quit Program
Offer to Save Changes
«extend»
«extend»
«include»
«include»
«extend»
User

ตัวอย่าง Use Case Description – Add a Person

- Add a Person เริ่มเมื่อผู้ใช้กดปุ่ม “Add” บนหน้าต่าง
- Diaglog จะปรากฏเพื่อให้ผู้ใช้กรอกชื่อ นามสกุล และข้อมูลต่าง ๆ
- ผู้ใช้สามารถ
  - กดปุ่ม OK Diaglog จะปิดไป แล้วระบบจะสร้างข้อมูลบุคคลเพิ่มขึ้นต่อท้าย Address Book และเพิ่มชื่อบุคคลนั้นที่ท้ายลิสต์ของหน้าต่างหลัก
  - กดปุ่ม Cancel Dialog จะปิดไปและไม่มีการเปลี่ยนแปลงข้อมูล

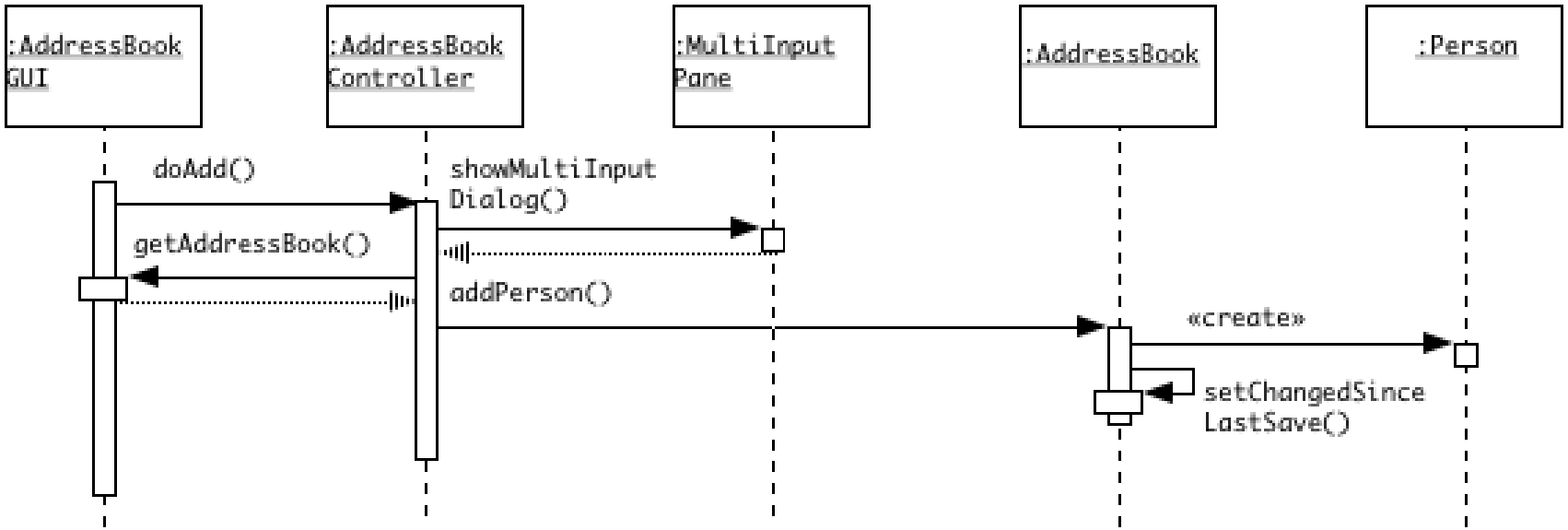
Analysis

- Entity แทน Address Book ที่โปรแกรมเก็บข้อมูล 1 ชิ้นในเวลาหนึ่ง ๆ
- Entity แทนข้อมูลบุคคลใน Address Book จำนวนไม่ระบุ
- GUI ซึ่งเป็น Boundary Object ระหว่างโปรแกรมและผู้ใช้
- Boundary Object ระหว่างโปรแกรมกับไฟล์
- Controller Object สำหรับแปลคำสั่งจากUI เพื่อทำงานของระบบ

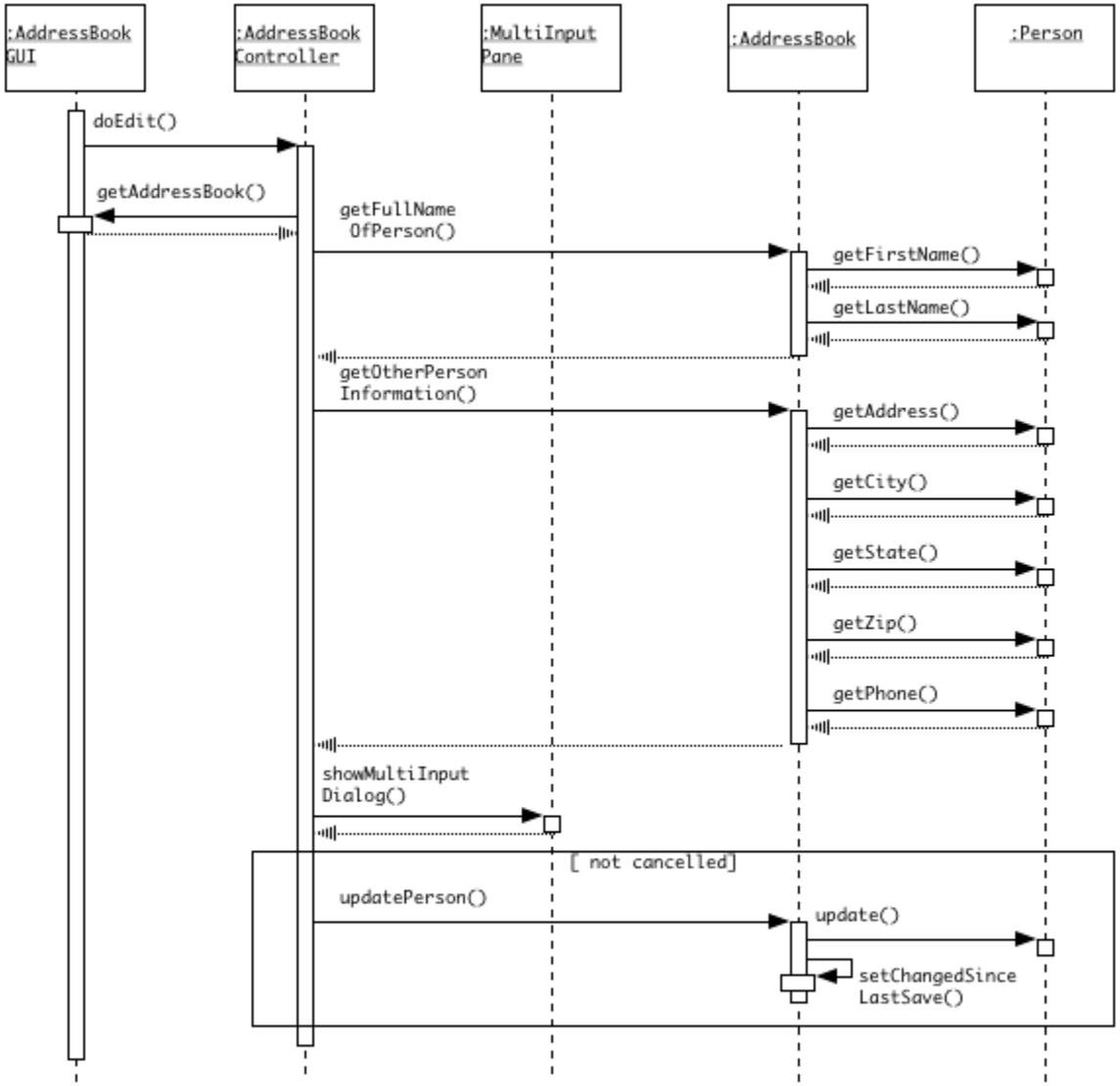


Sequence Diagram

Add a Person Use Case Sequence Diagram

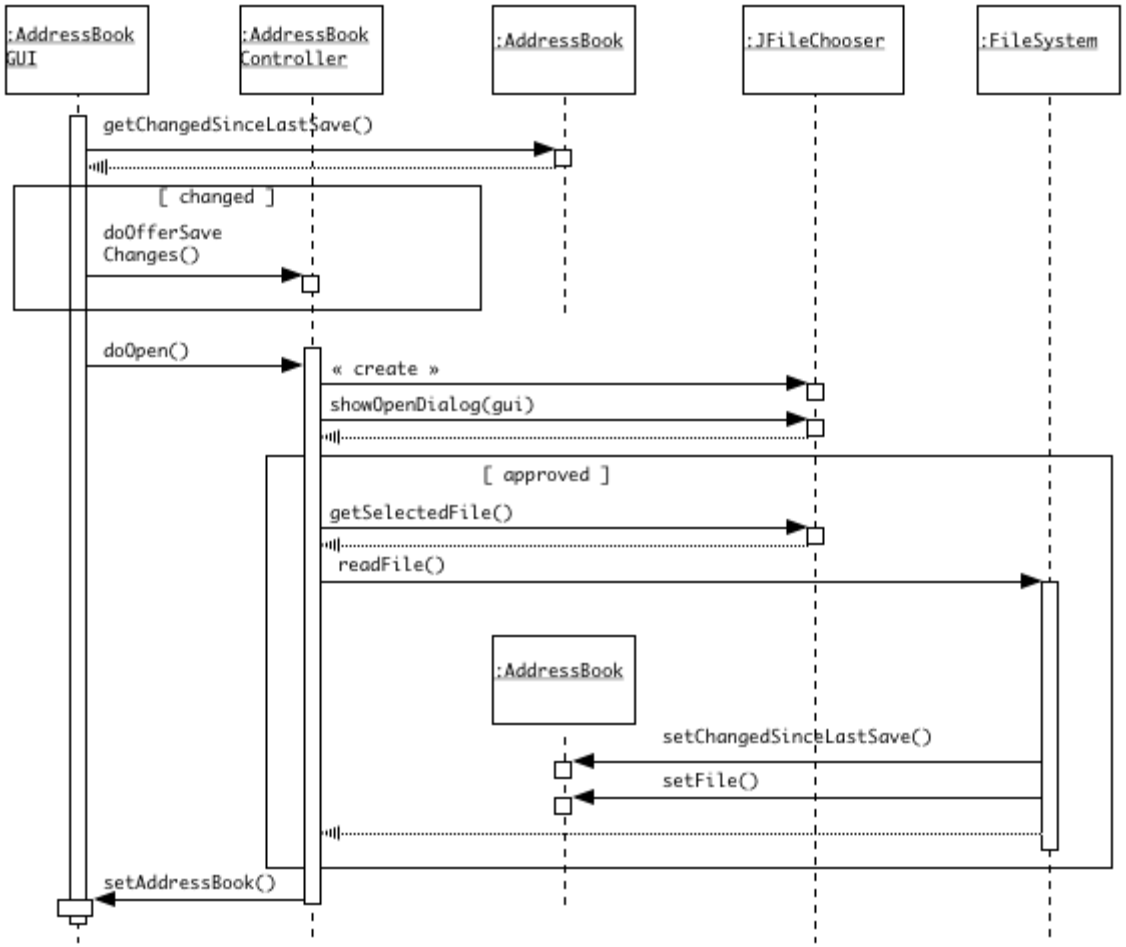


Edit a Person Use Case Sequence Diagram

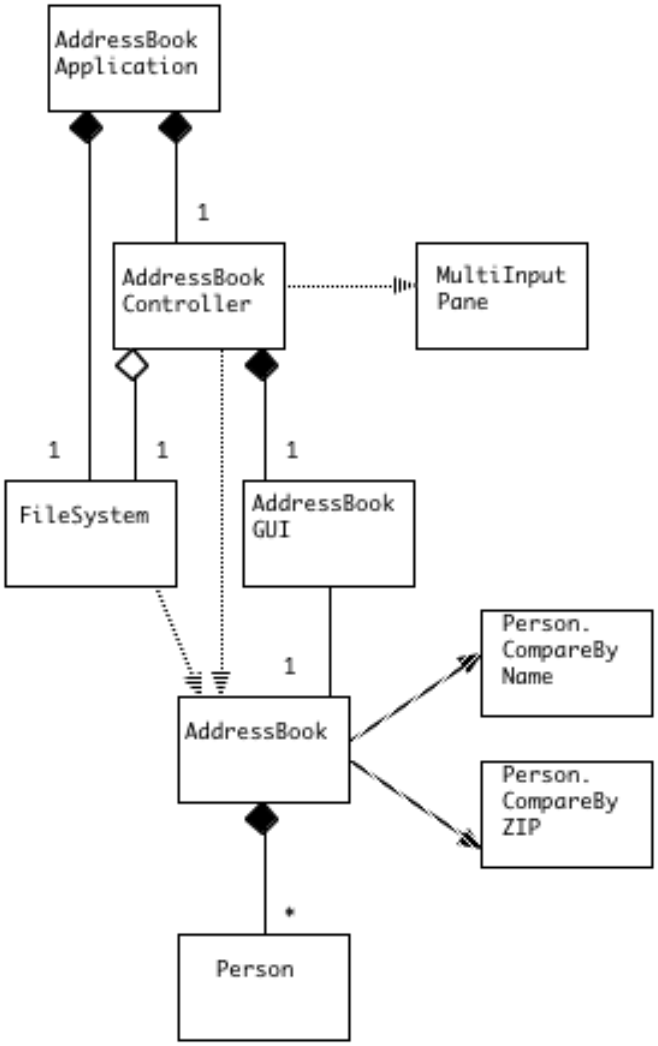


If there is no selected name, none of the above is done; instead, an error is reported

Open Existing Address Book Use Case Sequence Diagram



Class Diagram



Example Class Detail

AddressBook

```
- collection: Person [] or Vector
- count: int (only if an array is used for collection)
- file: File
- changedSinceLastSave: boolean
```

```
+ AddressBook()
+ getNumberOfPersons(): int
+ addPerson(String firstName, String lastName, String address,
            String city, String state, String zip, String phone)
+ getFullNameOfPerson(int index): String
+ getOtherPersonInformation(int index): String[]
+ updatePerson(int index, String address, String city,
               String state, String zip, String phone)
+ removePerson(int index)
+ sortByName()
+ sortByZip()
+ printAll()
+ getFile(): File
+ getTitle(): String
+ setFile(File file)
+ getChangedSinceLastSave(): boolean
+ setChangedSinceLastSave(boolean changedSinceLastSave)
```

AddressBookGUI

```
- controller: AddressBookController
- addressBook: AddressBook
- nameListModel: AbstractListModel
- nameList: JList
- addButton: JButton
- editButton: JButton
- deleteButton: JButton
- sortByNameButton: JButton
- sortByZipButton: JButton
- newItem: JMenuItem
- openItem: JMenuItem
- saveItem: JMenuItem
- saveAsItem: JMenuItem
- printItem: JMenuItem
- quitItem: JMenuItem
```

```
+ AddressBookGUI(AddressBookController controller,
      AddressBook addressBook)
+ getAddressBook(): AddressBook
+ setAddressBook(AddressBook addressBook)
+ reportError(String message)
+ update(Observable o, Object arg)
```

Class ที่มีลักษณะที่ไม่ดี

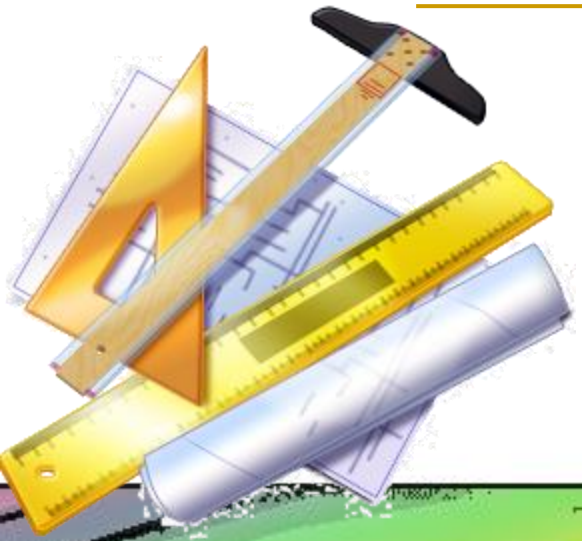
- Class ที่ใช้ชื่อไม่สื่อความหมายหรือสื่อผิด
- Class ที่ทำหน้าที่หลายอย่างเกินไป ให้สร้าง Class อื่นมาช่วย
- Class ที่เปลี่ยนแปลง class อื่น
- Class ที่ไม่ทำหน้าที่อะไร แต่มีอยู่เพียงเพื่อให้มี
- Class ที่มี features ที่ไม่ได้ถูกใช้งาน
- Class ที่มีความรับผิดชอบที่ไม่เกี่ยวข้องกัน
- Class ที่มีการสืบทอดแบบผิด ๆ เช่น ไม่ได้มีความสัมพันธ์ในกันจริง หรือเป็น class ที่สืบทอดแบบไม่มีประโยชน์
- Class ที่มีการทำหน้าที่ซ้ำไปซ้ำมา

จุดมุ่งหมายของบทนี้

- ทบทวนหลักการออกแบบเชิงวัตถุ
- ทบทวนการระบุหาวัตถุจากปัญหา
- แสดงตัวอย่างการออกแบบเชิงวัตถุ

ทบทวนปลายภาค

(Final Review)



Lecture 13

เยาวดี เต็มธนาภัทร์

Yaowadee Temtanapat

วัตถุประสงค์ของการเรียนวันนี้

- เพื่อทบทวนเนื้อหาก่อนสอบปลายภาค
  - โครงสร้างแบบกลุ่ม (Collection): อาร์เรย์ และ ArrayList
  - การสืบทอด Interface และการสืบทอดคลาส
  - Polymorphism
  - การจับและโยนความผิดพลาด
  - Generic
  - API ที่ควรรู้จัก

โครงสร้างแบบกลุ่ม

- การประกาศตัวแปรและการสร้างวัตถุ
  - **อาร์เรย์**: เก็บได้ทั้งกลุ่มของวัตถุและข้อมูลชนิดพื้นฐาน

```
DataType[] variableName;
variableName = new DataType[size];
```

    - การเพิ่มสมาชิก กรณีชนิดพื้นฐาน (Primitive Data Type)

```
variableName[index]  = value;
```

    - การเพิ่มสมาชิก ต้องมีวัตถุเพื่อให้อ้างถึง (จึงอาจต้องสร้างวัตถุด้วย)

```
variableName[index]  = new DataType();
```

    - การหาขนาด

```
variableName.length
```

    - ต้องบริหารจัดการในเรื่องตำแหน่ง ขนาด และการเพิ่ม-ลดขนาดเอง

โครงสร้างแบบกลุ่ม

- การประกาศตัวแปรและการสร้างวัตถุ
  - **ArrayList**: เก็บกลุ่มของวัตถุเท่านั้น

```
ArrayList<DataType> variableName;
variableName = new ArrayList<DataType>();
```

    - การเพิ่มสมาชิก ต้องมีวัตถุเพื่อให้อ้างถึง (จึงอาจต้องสร้างวัตถุด้วย)

```
variableName.add(new DataType());
variableName.add(index, new DataType());
```

    - การลบสมาชิก

```
variableName.remove(index);
```

    - การหาขนาด

```
variableName.size();
```

Interface (1)

- การประกาศอินเทอร์เฟส

```
public interface InterfaceName {
    returnType methodName(Type param1);
    // ... ละลายเซ็นเมท็อดอื่น ๆ
}
```

- การสืบทอดอินเทอร์เฟส (Realization)
  - บังคับทำ Override ทุกเมท็อดของอินเทอร์เฟส

```
public class ClassName implements InterfaceName {
    public returnType methodName(Type param1) {
        // ... do something
        return something;
    }
    // implements เมท็อดอื่น
}
```

Interface (2)

- การแปลงชนิด

```
ClassName c1 = new ClassName();
InterfaceName in1 = c1;
InterfaceName in2 = new Interface();

ClassName c2 = in1;
ClassName c2 = (ClassName) in1;

in1.methodName(arg1);
c1.methodName(arg1);
```

Inheritance

- การประกาศคลาสสืบทอดจากอีกคลาส

```
public class SubClass extends SuperClass {
    // สืบทอดลักษณะและความสามารถของ SuperClass ยกเว้น constructor
    // implements เมท็อดหรือตัวแปรวัตถุอื่นเพิ่ม หากต้องการ
}
```

- การทำ Override

```
@Override
public returnType methodName() {
    // ... Implements การทำงานที่ต้องการ
    // หากต้องการใช้ความสามารถของ superclass ด้วย เรียก
    super.methodName();
}
```

Inheritance

- การแปลงชนิด

```
SuperClass sc = new SubClass();
SubClass sub = new SuperClass();
SubClass sub = (SubClass)sc;
```

- ควรตรวจชนิดก่อนการแปลงโดยใช้ instanceof

```
sc instanceof SuperClass
```

  - คืนจริง (true) หาก sc เป็นชนิด SuperClass

ดังนั้น

```
sc instanceof Object
```

  - คืน ???

Inheritance และ Constructor (1)

```
class Person {
    public void sayHello( ) {
       System.out.println("Hello");
    }
}
```

มีความหมายเท่ากับ

```
class Person {
    public Person( ) {
        super( );
    }
    public void sayHello( ) {
        System.out.println("Hello");
    }
}
```

Compiler เพิ่มให้โดยอัตโนมัติ

Inheritance และ Constructor (2)

- หากกำหนด Constructor โดยไม่เรียก super class constructor, compiler เพิ่มให้โดยอัตโนมัติ

```
class Student
      extends Person {
   private String name;

   public Student( ) {
        name = "unknown";
   }
}
```

```
class Student
      extends Person {
   private String name;

   public Student( ) {
        super();
        name = "unknown";
   }
}
```

ระวัง (Caution) !!

```
class Vehicle {
  private String vid;
  public Vehicle(String vNo)
  {
      vid = vNo;
  }

  public void getVid() {
      return vid;
  }
}
```

```
class Car extends Vehicle {
  private int noOfSeats;
  public void setSeat(int n)
  {
      noOfSeats = n;
  }

  public int getSeat() {
      return noOfSeats;
  }
}
```

Compilation Error เนื่องจากไม่มีการกำหนด constructor สำหรับ Car( ) compiler เพิ่ม `public Car() { super( );}` แต่ไม่มี constructor ที่ไม่รับพารามิเตอร์ใน superclass Vehicle

Polymorphism

```
class X { public int aMethod() { return 1; } }
class Y extends X { public int aMethod() { return 2; } }
class Z extends X { public int aMethod() { return 3; } }
public class Test {
   public static void main(String[] args) {
      X x1 = new X();
      X x2 = new Y();
      X x3 = new Z();
      System.out.println(x1.aMethod());
      System.out.println(x2.aMethod());
      System.out.println(x3.aMethod());
   }
}
```

โค้ดนี้พิมพ์อะไร???

การจัดการกับความผิดพลาด

- Exception Handling: ความผิดพลาดไม่ควรถูกละเลยและควรต้องได้รับการจัดการอย่างเหมาะสม
  - Method ส่งความผิดพลาดกลับไปยังผู้เรียก → throwing exception
    - อาจ**สร้าง**วัตถุแสดงความผิดพลาดแล้ว**ส่ง**ให้ผู้เรียก
    - อาจ**ส่งต่อ**ความผิดพลาดที่ได้รับจากการเรียก method อื่น
      - เมท็อดอื่นโยนความผิดพลาดมา ส่งต่อความผิดพลาดนั้นให้ผู้เรียก
  - Method สามารถจัดการความผิดพลาดได้เองอย่างเหมาะสม (ไม่ต้องส่งต่อจัดการเองได้ที่ method ที่ทำงานขณะนั้น) → catching exception
  - ผสมทั้ง 2 แบบเข้าด้วยกัน

การจัดการกับความผิดพลาด

- Checked Exception
  - ตรวจสอบโดย Compiler → ผู้พัฒนาต้องรับรู้เกี่ยวกับความผิดพลาดนั้น
    - ตัวอย่างเช่น IOException และ subclasses ของมัน เป็น checked exceptions
- Unchecked Exception
  - ไม่ตรวจสอบโดย Compiler
    - ทั่วไปเป็นความผิดพลาดในการเขียนของผู้พัฒนา
      - ตัวอย่างเช่น NullPointerException, ArrayIndexOutOfBoundsException, ClassCastException

การประกาศการส่ง/ส่งต่อ Exception

- Syntax ของการประกาศเมท็อดที่มีการส่งความผิดพลาด

```
accessSpecifier returnType methodName (Type variable,
   ...)
   throws ExceptionClass1, ExceptionClass2 . . .
```

- ตัวอย่างเช่น:

```
public void withdraw(double amt)
                throws OverdraftException
```

- จุดมุ่งหมาย:
  - เพื่อระบุเป็นข้อกำหนดให้รู้ว่า method นั้น ๆ มีการส่ง/โยนความผิดพลาด (โดยเฉพาะที่ต้องตรวจสอบในกรณี Checked Exception)

การจับความผิดพลาด

- try-catch-finally: เพื่อทำงานและจัดการกับ statements ที่อาจเกิดความผิดพลาดได้ และทำสิ่งจำเป็นเสมอ
  - กรณีไม่มีข้อผิดพลาด ทำเพียง statements ใน try block
  - กรณีมีความผิดพลาด หยุดการทำงานใน try เพื่อทำงานตาม statement ที่อยู่ใน catch clause ที่ใกล้เคียงที่สุด
    - หมายเหตุ ลำดับการจับ (catch) ความผิดพลาดต้องเรียงไล่จากเฉพาะที่สุดไปหาทั่วไป
  - เมื่อผ่านจากทั้ง 2 กรณีจะทำประโยค finally ก่อนกลับไปยังผู้เรียกเสมอ

- Syntax:

```
try {
    statement;
    //...
}
catch (ExceptionClass1
   exceptionObject) {
    statement;
    //...
}
catch (ExceptionClass2
   exceptionObject) {
    statement;
    //...
}
finally {
    statement;
    //...
}
```

การออกแบบวัตถุแสดงความผิดพลาดของเราเอง (1)

- สร้าง Class ที่สืบทอดจาก Exception หรือ Throwable
  - หากต้องการ Unchecked exception ให้สืบทอดจาก RuntimeException
    - ข้อดี เปิดทางให้ผู้พัฒนาอาจเลือกตรวจสอบ
    - ข้อด้อย อาจละเลยการตรวจสอบได้
- กำหนด Constructors 2 แบบ:
  - ไม่มี parameter ()
  - มี String เป็น parameter สำหรับแสดงเหตุผล (String reason)

การออกแบบวัตถุแสดงความผิดพลาดของเราเอง

- ตัวอย่าง Checked Exception

```
public class MyException extends Exception {
   public MyException() { super("Some exception reason!!"); }
   public MyException(String reason) { super(reason);  }
}
```

- ตัวอย่าง Unchecked Exception

```
public class AnotherException extends RuntimeException {
   public AnotherException() { super("Some reason!"); }
   public AnotherException(String reason) { super(reason); }
}
```

คลาสเจเนอริก Room<T>

- ประกาศคลาสเจเนอริก Room

```
public class Room<T> {
    private T object;

    public Room(T object) {
        this.object = object;
    }
    public T getObject() {
        return object;
    }
}
```

Syntax: ประกาศคลาสเจเนอริก

- Syntax:

  ```
  public class ClassName<GenericTypeList> { ... }
  ```

  เมื่อ GenericTypeList เป็นชื่อพารามิเตอร์ ถ้ามากกว่า 1 ตัวคั่นด้วย comma ,

- ตัวอย่างเช่น:

ชนิดพารามิเตอร์ตัวเดียว <T>

```
public class Room<T> {
   private T object;
   // ...
}
```

ชนิดพารามิเตอร์หลายตัว <T, S>

```
public class Room<T, S> {
   private T creature;
   private S treasure;
   //...
}
```

- จุดมุ่งหมาย เพื่อประกาศคลาสเจเนอริก ซึ่งเมื่อนำไปใช้กำหนด**ชนิดวัตถุ**ที่แท้จริง (*หมายเหตุ:* ไม่สามารถใช้กับชนิดพื้นฐานได้)

ตัวอย่างการใช้งานคลาสเจเนอริก ArrayList<E>

- การใช้งาน ทำเช่นเดียวกับคลาสอื่น แต่ต้องกำหนดชนิดที่แท้จริงที่จะทำงานด้วย
- การประกาศตัวแปรและการสร้างวัตถุสำหรับเป็นที่เก็บเหรียญ
  - ประกาศตัวแปร coinList ชนิด ArrayList<Coin>

    ```
    ArrayList<Coin> coinList;
    ```

  - สร้างวัตถุ ArrayList สำหรับเก็บเหรียญ และให้อ้างถึงโดยตัวแปร coinList

    ```
    coinList = new ArrayList<Coin>();
    ```

- การเรียกใช้เมท็อดต่าง ๆ ของวัตถุ ArrayList<Coin>
  - การใช้งานเพื่อเพิ่มเหรียญใน coinList

    ```
    coinList.add(new Coin("10 bahts", 10.0);
    ```

  - การใช้งานเพื่อดึงเหรียญลำดับที่ 0 ใน coinList

    ```
    Coin coin = coinList.get(0);
    ```

เมท็อดเจเนอริก

- กำหนดเมท็อดให้เป็นเจเนอริกได้เช่นเดียวกับคลาส
- Syntax:

```
modifier <T1,…, Tn> returnType methodName(parameterList) {
    // ตัวแปรท้องถิ่นและประโยคการทำงานของเมท็อด
}
```

  - T_i เป็นชนิดพารามิเตอร์ที่เมท็อดใช้เป็นชนิดของพารามิเตอร์เข้าของเมท็อด
  - parameterList เป็นรายการพารามิเตอร์ประกาศเช่นเดียวกับเมท็อดทั่วไป
- หมายเหตุ กรณีที่เมท็อดอยู่ในคลาสเจเนอริก และชนิดพารามิเตอร์เป็นของคลาส ไม่จำเป็นต้องระบุ <T_i> ที่อยู่หน้า returnType

ชนิดพารามิเตอร์แบบมีขอบเขต (Bounded Type Parameters)

- สามารถกำหนดชนิดพารามิเตอร์ให้มีขอบเขตว่า ต้องเป็น subclass ของบางชนิด
- ปัญหา: ต้องการให้ห้องมีวัตถุเฉพาะที่เป็นประเภทสัตว์ประหลาด (Monster และ subclass ของมัน) สามารถกำหนดขอบเขต

```
public class Room<T extends Monster> {
    // รายละเอียดที่เหลือเช่นเดียวกับ Room เดิม ในหน้า 8
}
```

- การใช้งานก็จะอยู่ในขอบเขตของกลุ่มประเภท Monster เท่านั้น
  - Room ไม่สามารถใช้กับชนิด Coin ได้อีกต่อไป

อินเทอร์เฟสเจเนอริก

- เช่นเดียวกับคลาส สามารถกำหนดอินเทอร์เฟสที่เป็นเจเนอริกได้
- อินเทอร์เฟสเจเนอริก (Generic Interface) ในจาวา API

```
public interface Comparable<T> {
    int compareTo(T obj);
}
```

  - โดยเมท็อด compareTo คืนค่า
    - ค่า 0 เมื่อวัตถุปัจจุบันมีค่าเท่ากับ obj
    - ค่าลบเมื่อวัตถุปัจจุบันมีค่าน้อยกว่า obj
    - ค่าบวก เมื่อวัตถุปัจจุบันมีค่ามากกว่า obj

API (1)

- Calendar, GregorianCalendar
  - เมท็อด: constructor, get
  - ใช้งานร่วมกับ SimpleDateFormat(pattern)
- Math
  - เมท็อด: sqrt, abs, ceil, pow, sin, cos, tan
- Random
  - เมท็อด: constructor, nextInt, nextDouble
- Scanner (ทั้งกรณีทำงานกับ Console และกับไฟล์)
  - เมท็อด: constructor, hasNext, nextInt, nextDouble, nextLine

API (2)

- String
  - เมท็อด: charAt, equals, equalsIgnoreCase, compareTo
- StringBuffer
  - เมท็อด: constructor, charAt, setCharAt, append, insert, deleteCharAt
- StringTokenizer
  - เมท็อด: constructor, hasMoreTokens, nextToken, countTokens
- อื่น ๆ: Color, Font