Introduction to Object

Vittayasak Rujivorakul

Lecture 2

Introduction to Object

Vittayasak Rujivorakul

Lecture 2

Outline

- แนะนำการโปรแกรมภาษา Java
- แนะนำแนวคิดเกี่ยวกับ Object Oriented
  - Encapsulation
  - Inheritance
  - Polymorphism

Java Programming Language

- **Java** คือ อะไร
  - เป็นโปรแกรมภาษาระดับสูง (High-level Programming Language)
  - เป็นเครื่องมือสำหรับการพัฒนา (Development environment)
  - เป็นเครื่องมือสำหรับการทำ Application (Application Environment)
  - เป็นเครื่องมือเพื่อการใช้งาน (Deployment Environment)
- ไวยกรณ์ (Java Syntax) คล้าย C++ แต่ความหมายคล้าย Smalltalk
- ใช้ในการพัฒนา Application และ Applet

Java Application

- สามารถนำภาษาจาวามาใช้ในการพัฒนาโปรแกรมได้เหมือนกับ โปรแกรมภาษาอื่นๆ เช่น Pascal, C, etc.
- หากแต่ Java จะถูก Compile ไว้ในลักษณะที่เป็น Byte Code ดังนั้นจะสามารถ run ได้ในเครื่องที่มี Java Runtime Environment เท่านั้น
- ดังนั้น การทำงานจึงไม่ขึ้นอยู่กับชนิดของเครื่องที่ใช้ run (platform-independence)
- ลักษณะของ Byte Code นั้นคล้ายกับ machine language แตกต่างที่ code ที่ได้นั้นจะเหมือนกันหมดในทุกๆ platform

Applet คืออะไร

- **Applet =**
  - โปรแกรมที่เขียนขึ้นโดยใช้ภาษาจาวา และถูกเก็บไว้ใน WWW Servers สามารถที่จะถูก download จาก Internet โดยใช้ Web Browsers เพื่อให้ทำงานได้อย่างปลอดภัยในเครื่อง client
- **ลักษณะ**
  - ปกติจะเป็นโปรแกรมขนาดเล็ก เพื่อให้ง่ายและรวดเร็วในการ download
  - ถูกเรียกผ่าน HTML web page เท่านั้น

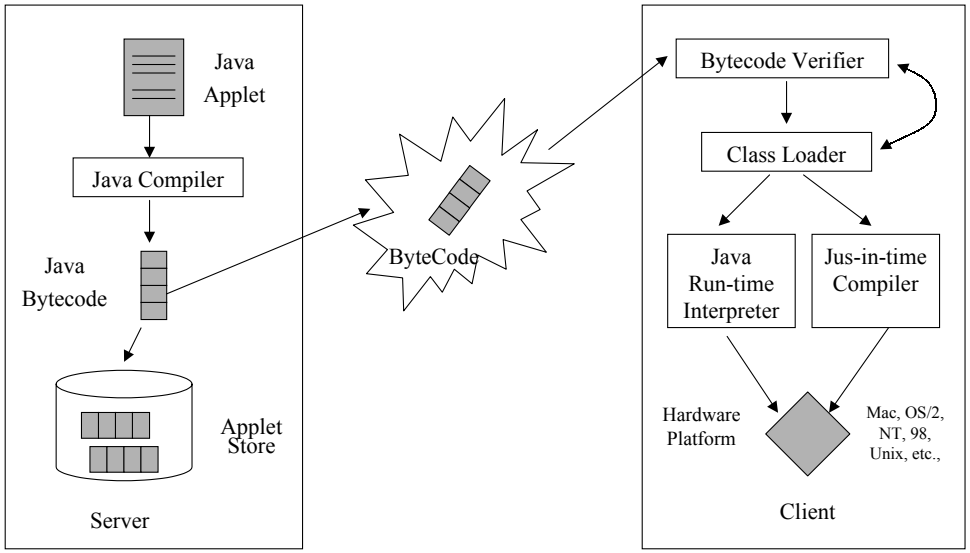
ลักษณะของภาษาจาวา

- **เป็นภาษาที่ง่าย**
  - ทั้งในด้านการอ่านและเขียน code
  - การจองหรือคืนพื้นที่ในหน่วยความจำเป็นไปโดยอัตโนมัติ
- **เป็นภาษา Object-Oriented**
  - object ประกอบด้วย 2 ส่วน: fields (บอกลักษณะของ object) และ methods (บอกความสามารถในการทำงานของ object)
  - ข้อดีของ OO programming:
    - ง่ายในการอ่านโปรแกรม
    - รวดเร็วในการพัฒนา
    - ให้ประสิทธิภาพในแง่การใช้ซ้ำ (reuse code)
    - มีความเสถียรสูงและข้อผิดพลาดต่ำ

ลักษณะของภาษาจาวา

- เป็นภาษาที่เพิ่มประสิทธิภาพของการพัฒนาโปรแกรม
- เป็นภาษาที่ Platform-independence
  - Code portability: ความสามารถในการย้ายไปทำงานได้ในหลายๆ Platform โดยไม่ต้องทำการแก้ไข code
- เป็นภาษาที่เป็น multi-threaded โดยกำเนิด
  - โปรแกรมจาวา แต่ละอันประกอบด้วยหลาย threads ซึ่งทำงานอย่างต่อเนื่องและเป็นอิสระต่อกัน
- เป็นภาษาที่รองรับการเปลี่ยนอย่าง Dynamic ในขณะ runtime
- เป็นภาษาที่มีมาตราการระบบความปลอดภัยที่ดี

หลักการทำงานของ ByteCode



องค์ประกอบของภาษาจาวา

- Java Virtual Machine(JVM)
- ตัวกำจัดขยะ (Garbage Collection)
- Code Security

JVM

- "imaginary" machine ที่ถูก implemented โดยการทำ S/W emulate บนเครื่องคอมพิวเตอร์จริง
- การทำงานรองรับกับ hardware platform specfication
- อ่าน byte code ที่ถูก compiled แล้ว
- สามารถ implement ได้ทั้งในรูปของ Software และ Hardware
- สามารถ implement ได้ในเครื่องมือที่ใช้ในการพัฒนาโดยใช้ Java Technology หรือ Web Browser

Garbage Collection

- **Deallocate memory ส่วนที่ไม่ได้มีการอ้างถึงแล้ว**
- **จาวาทำ garbage collection โดยอัตโนมัติ**
- **การทำงานแตกต่างกันไปขึ้นกับ JVM**

Code Security

- การตรวจสอบความปลอดภัยของ Code
  - **Loading Code** - ถูกจัดการโดย classes loader

    (แยก namespaces ของ local classes กับ remote classes, วาง memory layout ของ file)
  - **Verifying Code** - ถูกจัดการโดย bytecode verifier

    (ตรวจสอบ code 4 รอบว่าไม่ violate สิทธิไม่เปลี่ยนชนิดของ object ไม่ก่อให้เกิด stack overflow underflow, etc)
  - **Exeuting Code** - ถูกจัดการโดย runtime interpreter

    (ทำการ execute)

ลักษณะ (Characteristics) ของ O-O Programming

- ทุกอย่างในโปรแกรมคือ and object.
- โปรแกรมประกอบจากกลุ่มของ object ซึ่งบอกให้ object ทำงานโดยการส่ง messages
- แต่ละ object จะมีส่วนของ memory ของมันเองที่อาจสร้างจาก object อื่นๆ
- แต่ละ object มีชนิด (type)
- ทุก objects ที่เป็น type ชนิดเดียวกันสามารถที่จะรับ message แบบเดียวกันได้

แนวคิดที่เป็นพื้นฐานของ OO

- **Encapsulation:** การซ่อนส่วนรายละเอียดทุกอย่างของการ implementation ทั้งในแง่ลักษณะ (states) และ พฤติกรรม (behaviors) ของ object โดยกำหนดไว้เพียงส่วนของ interface เพื่อให้ผู้ใช้ภายนอกเรียกได้
- **Inheritance:** วิธีการในการสร้าง clone object โดยเพิ่มส่วนของ ลักษณะพิเศษที่แตกต่าง (some new features)
- **Polymorphism:** วิธีการในการทำให้ operation ชื่อเดียวกันมีพฤติกรรมการทำงานแตกต่างกันออกไปสำหรับ objects ชนิดที่ต่างกัน

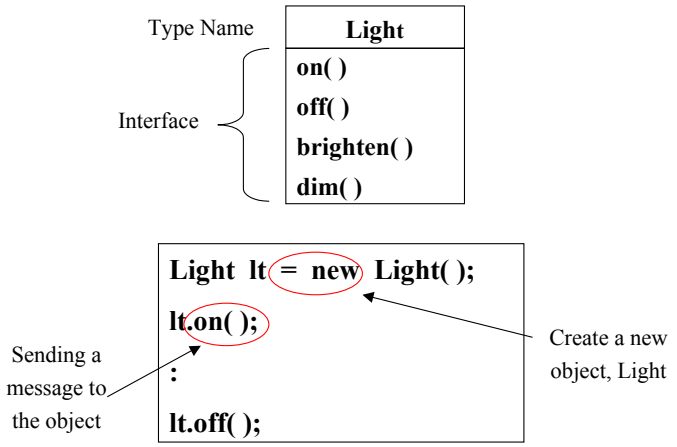
Object

- Object = สิ่งของที่เราสนใจ ทั้งที่จับต้องได้และไม่ได้
- Object ประกอบด้วยข้อมูลและ operations ที่ใช้จัดการกับข้อมูลนั้น
- การเขียนโปรแกรมในลักษณะ object-oriented style คือการนำ interacting objects มาประกอบกัน
  - ตัวอย่างเช่น โปรแกรมที่ใช้ในการจัดการกับ Bank Account สำหรับธนาคาร อาจประกอบจากหลาย Account, Customer, Transaction และ ATM objects
  - Account object ประกอบด้วย ข้อมูล เลขที่, ชื่อ, วันที่เปิด, balance เริ่มต้น, balance ปัจจุบัน และ operations เช่น deposit, withdrawal, transfer, etc.

Object มี interface

- แต่ละ entity คือ object
- แต่ละ object อยู่ใน class เฉพาะที่กำหนดลักษณะ (characteristics) และ พฤติกรรม (behaviors)
- ในการ program, เราสร้างแม่แบบ (template) ของ objects ที่เรียกในจาวาว่า class (อาจคิดว่าเป็นเสมือน type ในภาษา procedural อื่นๆ)
- วิธีในการร้องขอไปยัง object เพื่อที่จะให้ object ทำงานที่เป็นประโยชน์ได้จะถูกกำหนดโดยส่วนที่เรียกว่า interface

Example: Object



Implementation Hiding

- เราสามารถที่จะร้องขอไปยัง particular object ผ่าน interface
- การทำงานของ object ตามที่ร้องขอให้เสร็จสมบูรณ์นั้นเกี่ยวข้องกับการ coding หรือ implementation ร่วมไปกับการซ่อนส่วนของ data
  - เหตุผล 2 อย่างในการควบคุมการใช้ (access) data members คือ
    - เพื่อให้ง่ายในการแบ่งส่วนของ code ที่ไม่ต้องการให้ client programmer แตะต้อง เช่น ง่ายในการซ่อนส่วนภายในที่ผู้ใช้ไม่ควรจะต้องรับรู้หรือเกี่ยวข้อง
    - เพื่องให้ง่ายในการบังคับพฤติกรรมเฉพาะบางอย่าง และสะดวกต่อผู้ออกแบบในการเปลี่ยนแปลงภายในโดยไม่ส่งผลกระทบกับ client programmer

การระบุการใช้ (Access Specifiers)

- Explicit Keywords 3 ตัวที่ใช้ในการกำหนดขอบเขตการใช้งานใน class
  - **public:** อนุญาตทุกผู้ใช้
  - **private:** อนุญาตสำหรับการใช้ภายใน class เพียงอย่างเดียว
  - **protected:** อนุญาตสำหรับ class ลูกที่สืบทอดจาก class แม่
- Implicit Keyword, “friendly”, ในกรณีที่ไม่มีการระบุใดๆ
  - **friendly:** อนุญาตให้ใช้เพียงใน package เดียวกัน (package access)

การใช้ซ้ำ (Reusing the implementation)

- **Composition** หรือ ความสำพันธ์แบบ “has-a”: เป็นการสร้าง member object ของ class หนึ่งภายในอีก class หนึ่ง (เช่น class ของ car จะมี class ของ engine เป็นส่วนประกอบ)
- **ข้อแนะนำ** สำหรับนักพัฒนามือใหม่: ไม่แนะนำให้พยายามทำการ over-reuse a class โดยวิธีการสืบทอด (inheritance) มากเกินไป, ควรที่จะพิจารณาเลือกใช้ composition ซึ่งเป็นวิธีที่ง่าน ยืดหยุ่นกว่า ในการสร้าง class ใหม่

Inheritance

- Inheritance เป็นวิธีการในการ clone พร้อมกับทำการเพิ่มหรือเปลี่ยนแปลงลักษณะของ class ที่มีอยู่เดิม (class ที่มีอยู่เดิมถูกเรียกว่า **base** class และ class ที่สืบทอดใหม่นั้นจะถูกเรียกว่า derived class
- Inheritance ถูก implemented ใน Java โดยการใช้ **extends** keyword
- ทั้ง base และ **derived** class จะมี interface แบบเดียวกัน
  - ในกรณีที่ไม่มีการเปลี่ยนแปลงใน derived class สำหรับ interface นั้นๆ, base-class interface จะสืบทอดลงสู่ derived class (ให้พฤติกรรมแบบเดียวกัน)
  - ในกรณีที่มีการสร้าง functions ใหม่ใน derived class, base class จะไม่รับรู้เกี่ยวกับ interface ใหม่นั้น นั่นคือการสืบทอดเป็นไปในทิศทางเดียว

Overriding base-class functionality

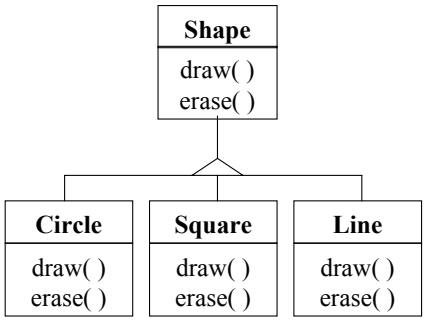
- ถ้าการทำการ change the behavior ของ function ที่มีอยู่ใน base-class, เรากล่าวว่าเป็นการทำการ **overriding** functions นั้น
- การ override a function ทำได้โดยการสร้าง definition ใหม่สำหรับ function นั้นใน derived class

เราต้องการเรียกใช้ interface function ชื่อเดียวกันใน class สืบทอด,แต่ต้องการให้มันทำสิ่งที่แตกต่างออกไปใน class ใหม่นั้น

ความสัมพันธ์แบบ Is-a VS Is-like-a

- Pure-substitution: derived class ทำการ overrides เพียงเฉพาะ base-class functions ในกรณีเช่นนี้ถือว่าเป็นความสัมพันธ์แบบ **“Is-A”**
- หากแต่ในบางกรณี, เราต้องการไม่เพียงแต่จะขยายหรือ override ส่วนของ interface ที่มีอยู่เท่านั้นหากต้องการที่จะเพิ่ม interfaces ใหม่ด้วย ในกรณีเช่นนี้เราถือว่าเป็นความสัมพันธ์แบบ **“Is-Like-A”**

Example: Inheritance



สังเกตว่า โปรแกรมสามารถที่จะหา interface ที่ถูกต้องและเหมาะสม ได้ด้วยตัวมันเอง โดยไม่ต้องมีการระบุว่า object นั้นเป็นอะไร (ถ้าหากเป็น วงกลม ให้ทำแบบนี้, ถ้าเป็นสี่เหลี่ยมให้ทำแบบนี้ ฯลฯ)

ถ้าหากว่าเรามี function

```
void doStuff( Shape s ) {
    s.erase( );
    :
    s.draw( );
}
```

Type upcasting:

```
Circle c = new Circle( );
Square s = new Square( );
Line l = new Line( );
doStuff( c );
doStuff( s );
doStuff( l );
```

Dynamic Binding

- เมื่อส่ง message doStuff(), เราไม่จำเป็นที่จะต้องรู้ว่าจะต้องจัดการกับรูปร่างแต่ละอันอย่างไร Java compiler และ run-time system จะทำการจัดการรายละเอียดเหล่านี้เอง กระบวนการจับคู่ให้ type กับ method ที่ตรงกันกับ type นั้นในช่วง runtime ถูกเรียกว่า **dynamic binding**
- ในกลไกนี้ เราสามารถที่จะส่ง message เดียวกันไปยัง object ต่างชนิดกัน แล้วสามารถที่จะเลือกและทำงานที่ถูกต้องได้นั้นผ่านวิธีการที่เรียกว่า **polymorphism** (operation อย่างเดียวกันสามารถที่จะแสดงพฤติกรรมที่ต่างกันได้สำหรับ object ต่างชนิดกัน)

Abstract Class

- ในการ design ที่ต้องการเพียง base class เพื่อกำหนดเพียง interface สำหรับ derived class, เราสามารถทำได้โดยการสร้าง **abstract class**
- เมื่อใช้ abstract, จะเป็นการบอกโดยนัยสำหรับผู้ใช้ว่า

  "นี่คือ interface function เพื่อให้ใช้ในการสืบทอดจาก class นี้ได้, แต่ว่าตัว class มันเองยังไม่ได้มีส่วนของการ implementation ใดๆ ไว้ให้"
- ดังนั้นเราไม่สามารถสร้าง instance จาก abstract class ได้
- abstract method บังคับให้ผู้ใช้เป็นผู้กำหนดส่วนของ code ที่มีความหมายในการใช้งานด้วยตัวเอง

Exception Handling

- วิธีการที่จะจัดการกับความผิดพลาด (error หรือ exception) และเป็นการบังคับผู้ใช้ไม่ให้ละเลย error เหล่านี้อย่างกลายๆ
- ในจาวา, การจัดการกับ error นั้นผูกพันแน่นกับตัวภาษา ดังนั้นภาษาจาวาไม่ยอมให้ละเลย error เหล่านี้ นักพัฒนาต้องเขียน code ของการจัดการกับ error ที่อาจเกิดขึ้นเหล่านี้อย่างเหมาะสม
- Exception นั้นสามารถที่จะโยนจากจัดที่เกิด error นั้นและจะถูกจับ "caught" โดยตัวจัดการกับ error (exception handler) ที่เหมาะสม
- 2 ตัวเลือกในการจัดการ: ส่งต่อ (throw up) หรือ จับ (catch) errors

Multithreading

- เพื่อทำการจัดการกับ tasks มากกว่าหนึ่งอย่าง ณ เวลาใดเวลาหนึ่งโดยการแบ่งโปรแกรมออกเป็นส่วนย่อยที่จะทำงาน แต่ละส่วนย่อยที่ทำงานนี้เรียกว่า **thread**
- ภาษาจาวานี้สร้างลักษณะของ thread ในตัวภาษาเอง
- จาวาทำการจำกับดการ lock ทรัพยากร เพื่อช่วยในการทำการ synchronize threads หลายอันและเป็นการแบ่งการใช้ทรัพยากรส่วนร่วม โดยอาศัย **synchronized** keyword

การวิเคราะห์และการออกแบบ (Analysis & Design)

- กฎพื้นฐาน: พยายามทำการค้นหา
  - ว่าอะไรคือ object? ต้องทำการแบ่ง project ของเราอย่างไรให้เป็นส่วนย่อย (component parts)
  - อะไรบ้างที่เป็นส่วน interfaces ของ object เหล่านี้, messages อะไรบ้างที่ต้องการเพื่อที่จะทำให้สามารถติดต่อกันได้ระหว่าง objects

Java in action

- ทุกอย่างในภาษาจาวาคือ object
- อย่างไรก็ตาม ในการ manipulation จริงๆ ของ object นั้นจะผ่าน identifier ซึ่งจะทำหน้าที่เป็นตัวอ้างถึง (reference) object นั้น
  - การสร้างตัวจัดการกับ object (การประกาศตัวแปรเพื่อใช้ในการอ้างถึง)

  String s;
  - การสร้าง object ใหม่

  S = new String (“Hello”);
- ทั้งหมดนี้ยกเว้น data type ที่เป็น primitive type

ขอบเขต (Scoping)

- ขอบเขตของ variable ในจาวากำหนดโดยวงเล็บปีกกา { }

```
{
    int x = 12;   // only x is available
    {
        int  q = 96;  // both x and q are available
    }
}
{  int  x = 10;
     { int x = 8; //illegal }
}
```


การสร้าง data type อันใหม่: class

- การสร้าง class ใหม่ในจาวาทำได้โดย:

```
class ANewType{
        // class body goes here
}
```

- ภายใน class จะบรรจุด้วย Fields และ Methods สำหรับ class นั้น

```
class DataOnly{
      int  i;
      float  f;
      boolean  b;
}
```

Methods, arguments และการ return ค่า

- Syntax พื้นฐาน

```
returnType  methodName (/* argument list */) {
    /* method body */
    return ???;
}
```

- ในกรณีที่ไม่ต้องการส่งคืนค่าใดใช้ **void**
- ในการเรียกหรือการส่ง message ทำได้โดยอ้างชื่อ object นั้นตามด้วยจุดและตามด้วยชื่อ method พร้อมกับ arguments ที่เหมาะสม

```
returnType  x = a.methodName(y, z);
```


การเรียกใช้ components อื่น

- การนำ component อื่นเข้ามาใช้ **import** keyword

```
import  java.util.Vector;
import  java.util.*;
```

- โดย default, java.lang.* จะถูก imported เข้ามาโดยอัตโนมัติ