Programming Fundamentals II

Week 4: Classes and Objects

Modified by
Kanjana Eiamsaard
2021.12.27

Let’s recap the previous class

1. Scanner/Format Class
2. Arrays
3. Strings
4. ArrayLists
5. Looping over a collection
6. enum
7. HashMap

Let’s continue our journey…

Classes and Objects

1. The beginning of the Objects
2. Class member
    1. Attribute
    2. Method
3. Inner class & Static nested class
4. Exception
5. Garbage collection
6. UML Class diagram

ระบบนิเวศน์ (Ecosystem)

- ระบบนิเวศน์ คือ ความสัมพันธ์อย่างเป็นระบบที่อยู่ในสิ่งแวดล้อม โดยสิ่งที่อยู่ภายใต้ระบบเดียวกัน ทำหน้าที่ของตนเพื่อเกื้อหนุนให้กับสิ่งรอบข้าง ให้อยู่ร่วมกันได้ และบรรลุถึงวัตถุประสงค์ที่กำหนด
- ตัวอย่าง มหาวิทยาลัยถือเป็นระบบนิเวศน์หนึ่ง ที่ประกอบไปด้วย มนุษย์ สิ่งอำนวยความสะดวก กฎระเบียบ องค์ความรู้ สัตว์ ฯลฯ
  - สิ่งเหล่านี้คือ คำที่เป็นนามธรรม ที่ใช้เรียกรวมสิ่งต่าง ๆ ที่ทำหน้าที่เฉพาะของตน ให้บรรลุถึงวัตถุประสงค์ในการสร้างมหาวิทยาลัยขึ้น

ข้อดีของการนำแนวคิดเชิงวัตถุมาใช้ในการพัฒนาโปรแกรม

- นำกลับมาใช้ใหม่ได้ (Reuse)
- ประหยัดเวลาในการพัฒนา (Rapid Delivery)
- ใช้งานง่าย (User Friendly)
- ดูแลรักษาได้ง่าย (More Maintainable)
- มีคุณภาพสูง (Greater Quality System)

ถ้าเราทำให้ “โปรแกรมคือระบบนิเวศของวัตถุ”

- ทุกครั้งที่โปรแกรมถูกสร้างขึ้น มักมีวัตถุประสงค์ในการสร้างเสมอ
- เมื่อเรานำคำว่าระบบนิเวศน์มาเปรียบเทียบกับการพัฒนาโปรแกรมด้วยแนวคิดเชิงวัตถุแล้ว ย่อมหมายถึง

  “การทำงานร่วมกันของสิ่งต่าง ๆ ที่มีลักษณะและความสามารถเฉพาะตัว เพื่อให้บรรลุวัตถุประสงค์ในการสร้างโปรแกรม”
- ดังนั้นเราต้องทำเข้าใจความหมายและการนำ “สิ่งต่าง ๆ” ที่ได้กล่าวไว้ไปใช้ในการสร้างโปรแกรมอย่างถูกต้อง

วัตถุ (Objects)

- ก่อนหน้านี้เราพูดถึง “สิ่งต่าง ๆ” ที่เกิดขึ้นในการพัฒนาโปรแกรม ซึ่งสามารถเทียบได้กับคำว่า “วัตถุ” ในการพัฒนาโปรแกรมด้วยแนวคิดเชิงวัตถุ
- วัตถุ ในที่นี้ จึงหมายถึง สิ่งที่ประกอบไปด้วยคุณลักษณะ (Attribute) และความสามารถ (Method)
  - Attribute คือ สิ่งที่แสดงถึงลักษณะของวัตถุและสิ่งที่ทำให้วัตถุชนิดเดียวกันมีความแตกต่างกัน เช่น คุณลักษณะของอ็อบเจกต์รถยนต์ ประกอบด้วย ล้อ เครื่องยนต์ ระบบเกียร์ ความเลขตัวถัง

วัตถุ (ต่อ)

- ความสามารถของวัตถุ (Method) คือ สิ่งที่วัตถุสามารถทำได้ ซึ่งมักเป็นคำกริยา เช่น สอนหนังสือ ค้นคว้าความรู้ เห่า
- อีกสิ่งที่ขาดไม่ได้ในระบบนิเวศคือ “ความสัมพันธ์ของวัตถุ” สาเหตุเพราะระบบนิเวศไม่อาจสมบูรณ์ได้ หากปราศจากความร่วมมือของแต่ละหน่วยย่อย ดังนั้นการกำหนดความสัมพันธ์ระหว่างวัตถุจะช่วยให้การกำหนดขอบเขตการทำงานของวัตถุแต่ละประเภทได้ดียิ่งขึ้น

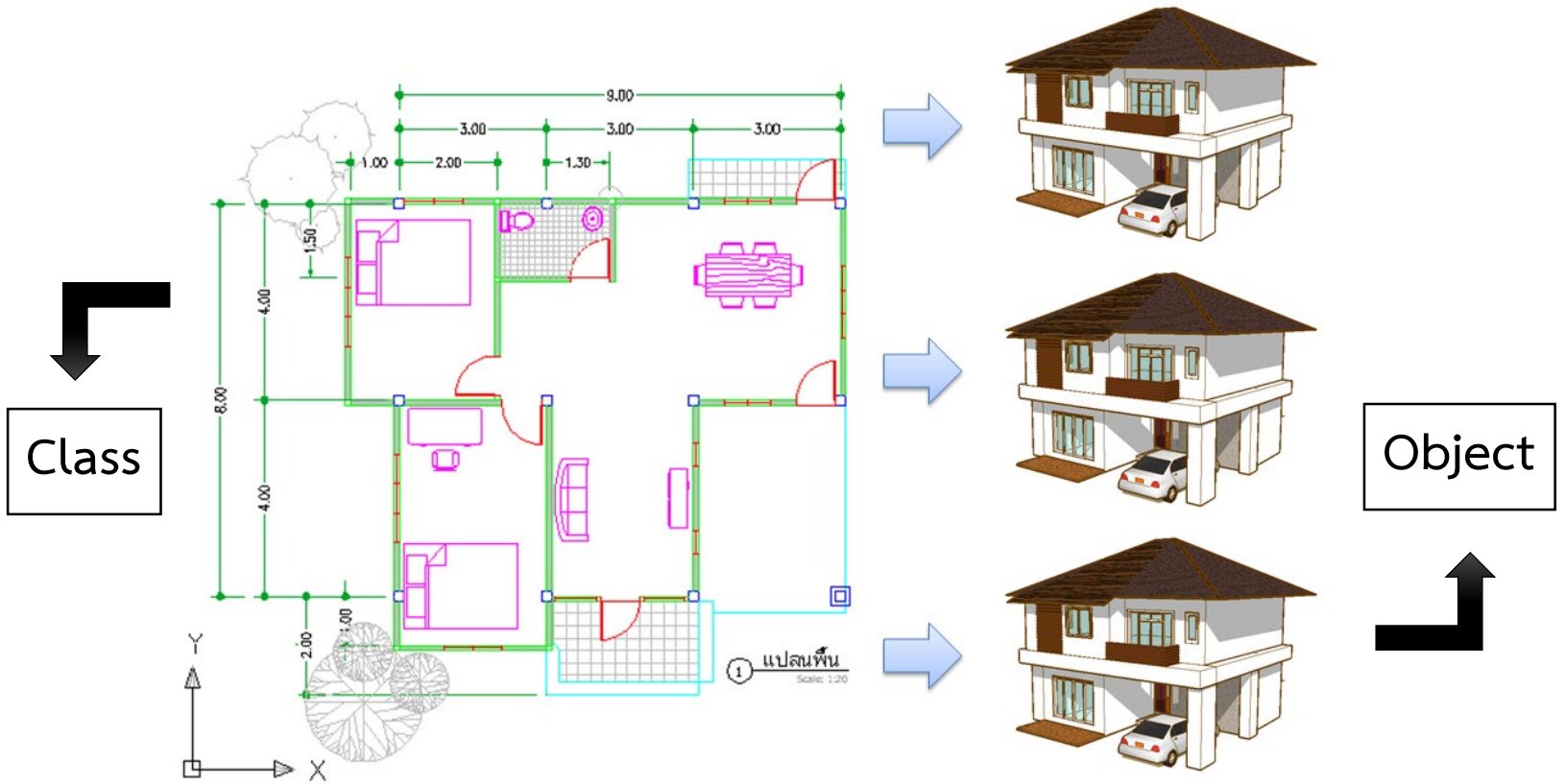
ตัวอย่างวัตถุในมหาวิทยาลัย

- อ.ปุณณะ อ.โก้ อ.ใหม่ อ.บอล อ.แอน
- อ.ประวิทย์ (หัวหน้าภาคคอม) อ.สถาพร (คณบดี วศ.ศรช) อ. อำนาจ (คณบดี วจก)
- คุณนก พี่ทราย พี่โก้ พี่เปรี้ยว พี่เก๋ น้องดรีม
- ไอ้เลอะเทอะ ไอ้ชาบู
- ลานเทเลทับบี้ โรงอาหารวิทยาเขต อาคารพละ สนามแบต
- เร้าท์เตอร์ตึก23 สายแลนหน้าห้อง309 งานวิจัยเรื่อง “The..”
- ฯลฯ

ความเกี่ยวเนื่องกันระหว่างคลาสและวัตถุ

- วัตถุ คือ ตัวตนของคลาส
- เราสามารถทำความเข้าใจกับคำว่า “คลาส” ผ่านลักษณะความเกี่ยวข้องระหว่างคลาสกับวัตถุได้ 2 รูปแบบ
  1) **คลาสเป็นพิมพ์เขียวของวัตถุ** คือ การเทียบเคียงกับกระบวนการผลิตในโลกความเป็นจริง เช่น รถยนต์ บ้าน สิ่งเหล่านี้มีจุดเริ่มต้นในการสร้างคือ พิมพ์เขียวที่ ซึ่งจะทำให้วัตถุที่เกิดขึ้นมีคุณลักษณะแบบเดียวกัน และความสามารถที่เท่าเทียมกัน

คลาสเป็นพิมพ์เขียวของวัตถุ



ความเกี่ยวเนื่องกันระหว่างคลาสและวัตถุ (ต่อ)

2) คลาสเป็นกลุ่มชนิดของวัตถุ กล่าวคือ หากพิจารณาตัวอย่างวัตถุที่เกิดในมหาวิทยาลัยแล้ว เราสามารถจำแนกกลุ่มของสิ่งเหล่านั้นได้ ตามลักษณะและความสามารถ

- เช่น กลุ่มมนุษย์ กลุ่มสิ่งของ สถานที่ ซึ่งชื่อเหล่านี้ไม่สามารถระบุตัวบุคคล/สิ่งเฉพาะ
- การแบ่งกลุ่มสามารถทำให้ละเอียดขึ้นได้ตามความเหมาะสมในการนำไปใช้งาน (ไม่กว้างเกินไปที่ขาดเอกลักษณะ ไม่แคบเกินไปจนไม่เหลือวัตถุในกลุ่ม)
- เช่น นิสิตปริญญาตรี นิสิตระดับสูงกว่าปริญญาตรี อาจารย์ เจ้าหน้าที่ ผู้บริหาร อุปกรณ์เครือข่าย อุปกรณ์การสอน งานวิจัย หนังสือ เป็นต้น

การประกาศคลาส

```
<class-modifiers> class ClassName {
    <field-declarations>
    <constructor-declarations>
    <method-declarations>
}
```

<class-modifiers> เป็นตัวกำหนดคุณสมบัติบางประการของคลาส เช่น **public, abstract, final** โดยในเบื้องต้นเราจะใช้ตัวกำหนด **public** หรือไม่ระบุ

<field-declarations> เป็นส่วนประกาศตัวแปรต่าง ๆ

<constructor-declarations> เป็นส่วนประกาศตัวสร้าง (**constructor**) ซึ่งเป็นเมทอดชนิดพิเศษสำหรับกำหนดสถานะเริ่มต้นให้กับอ็อบเจกต์

<method-declarations> เป็นส่วนประกาศเมทอดทั่วไป

ตัวอย่างการประกาศคลาส

```
class Juggler {
    private int value;

    public Juggler(int value) {
        this.value = value;
    }

    public void swapWith(Juggler another) {
        int temp = this.value;
        this.value = another.value;
        another.value = temp;
    }

    public int getValue() {
        return value;
    }
}
```

การสร้างอ็อบเจกต์

- ใช้คำสั่ง “new” ตามด้วยชื่อคลาสและอาร์กิวเมนต์ที่ต้องการส่งให้กับคอนสตรักเตอร์ของคลาส

```
ClassName varName = new ClassName(<argument-list>)
```

```
Juggler a = new Juggler(10);
Juggler b = new Juggler(20);
```

```
LocalDateTime time = LocalDateTime.now();
```

Attribute

ตัวแปรภายในคลาส

```
<class-modifiers> class ClassName {
    <field-declarations>
    <constructor-declarations>
    <method-declarations>
}
```

- ในส่วน <field-declarations> คือ พื้นที่สำหรับกำหนดตัวแปรภายในคลาส สิ่งที่ต้องทราบเกี่ยวกับตัวแปรคือ
- ความเป็นเจ้าของตัวแปร แบ่งออกเป็น 2 ชนิด
- ชนิดของตัวแปร แบ่งออกเป็น 2 ชนิด

ความเป็นเจ้าของตัวแปร

- พิจารณาจากสิทธิ์ในการควบคุมข้อมูลในตัวแปร ว่าสิทธิ์นั้นเป็นของ **คลาส หรือ วัตถุ**
- ตัวแปรของอินสแตนซ์ (instance variable) คือ ตัวแปรที่วัตถุเป็นเจ้าของ โดยที่ค่าของตัวแปรจะต่างกันเมื่ออยู่ต่างวัตถุกัน
- ตัวแปรของคลาส / ตัวแปรสถิต (class / static variable) คือ ตัวแปรของคลาส โดยที่ค่าของตัวแปรจะเหมือนกัน ถึงแม้จะอยู่ต่างวัตถุก็ตาม

การกำหนดค่าให้ตัวแปรของอินสแตนซ์

Instance variables are initialized in one of the four way

- By a default value (0, 0.0, false, or null)
- In constructors
- At declaration site
- In an initialization block

1. ตัวแปรจะถูกสร้าง และทำงานพร้อมกับการ new class เพื่อสร้าง object
2. ตัวแปรจะถูกเรียกใช้งานจาก method, constructor หรือ block ภายใน Class

```
2  public class Employee3
3  {
4          private String name;
5          private double salary;
6          private static int lastId = 0;
7          private int id;
8
9          // Constructor Method
10         public Employee3()          // Overload with duplicate
11         {
12             //Employee("",0.00);    //incorrect
13             this("",0.00);
14         }
```

ตัวแปรของคลาส / ตัวแปรสถิต

```java
public class Employee {
    private String name;
    private double salary;
    private final int id;
    private static int lastId = 1000;

    public static final double SALARY_STEP_SIZE = 10.0;

    public Employee(String name, double salary) {
        this.name = name;
        this.salary = computeNextSalaryStep(salary);
        id = ++lastId;
    }

    public static double computeNextSalaryStep(double salary) {
        // Round to the next salary step
        double steps = Math.ceil(salary / SALARY_STEP_SIZE);
        return steps * SALARY_STEP_SIZE;
    }

    public void raiseSalary(double percent) {
        double raise = salary * percent / 100.0;
        salary = computeNextSalaryStep(salary + raise);
    }

    public void showProfile() {
        System.out.printf("Name: %s%n", name);
        System.out.printf("ID: %d%n", id);
        System.out.printf("Salary: %,.2f%n", salary);
    }
}
```

```java
Employee george = new Employee("George", 15_233.00);
Employee sarah = new Employee("Sarah", 18_500.00);

george.showProfile();

sarah.raiseSalary(12.5);
sarah.showProfile();
```

```
Name: George
ID: 1001
Salary: 15,240.00
Name: Sarah
ID: 1002
Salary: 20,820.00
```

Static variables

```
public class Employee {
    private static int lastId = 0;
    private int id;
    ...
    public Employee() {
        lastId++;
        id = lastId;
    }
}
```

Share all This Class

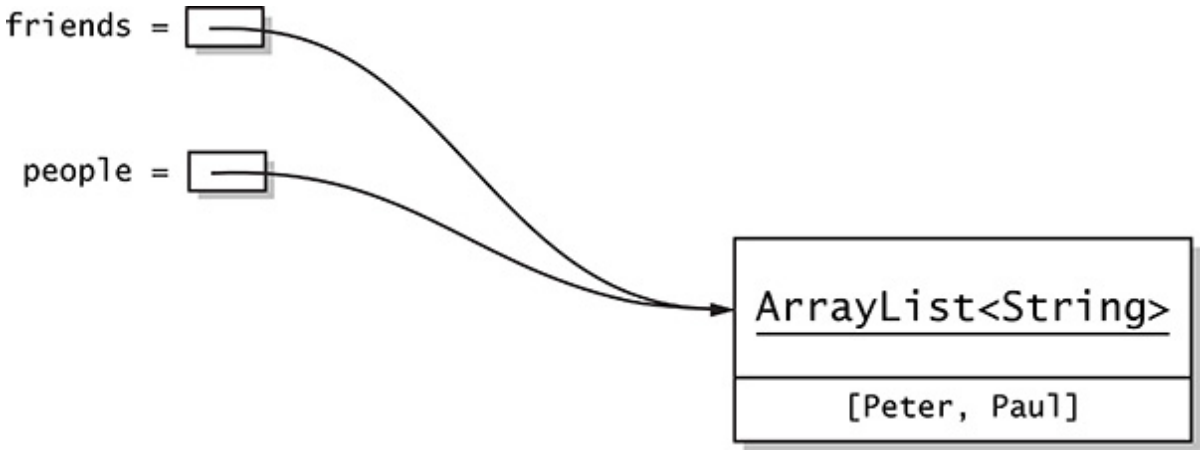
Every instance of the Employee class share lastId

ชนิดของตัวแปร

- ตัวแปรชนิดพื้นฐาน (Primitive type) คือ ตัวแปรที่เก็บข้อมูลไว้ที่ตัวแปรโดยตรง เช่น int, float, double ฯลฯ
- ตัวแปรอ้างอิง (Reference type) คือ ตัวแปรที่เก็บที่อยู่ของข้อมูล เช่น array, class, enum

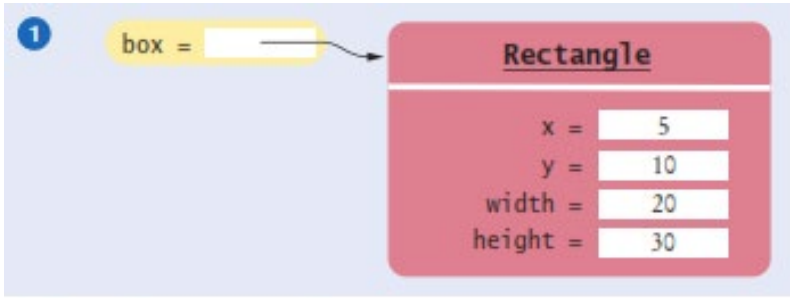
ตัวแปรอ้างอิงและการเกิดสมนาม (alias)

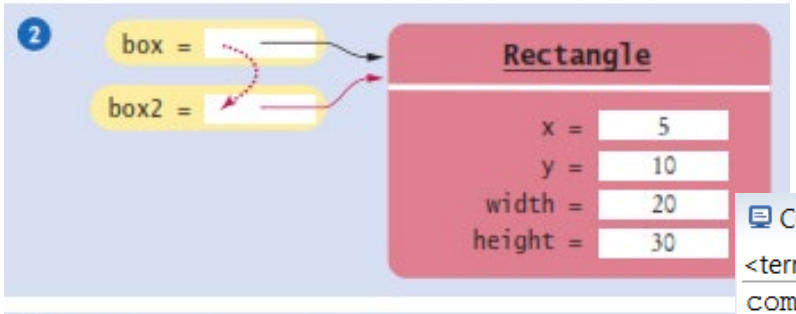
```
ArrayList<String> friends = new ArrayList<String>();
   // friends is empty
friends.add("Peter");
   // friends has size 1
ArrayList<String> people = friends;
   // Now people and friends refer to the same object
people.add("Paul");
```

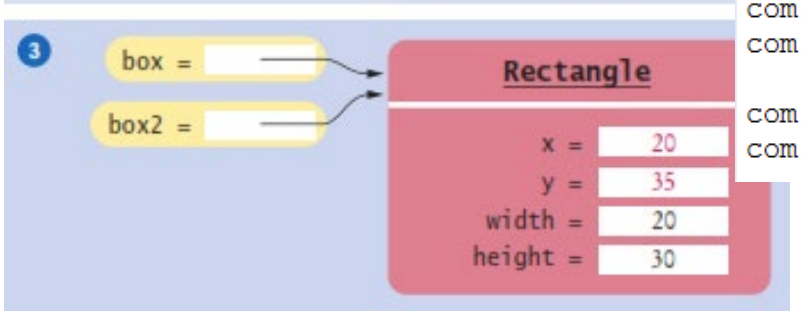


ตัวแปรอ้างอิงและการเกิดสมนาม (ต่อ)

```
Rectangle box = new Rectangle(5, 10, 20, 30);      // ———— (1)
Rectangle box2 = box;                              // ———— (2)
box2.translate(15, 25);                            // ———— (3)
```







Console | Problems | Javadoc | Declaration

<terminated> Study03ObjectReference2 [Java Application] C:\Program Files\Java\jdk1.8.0_1

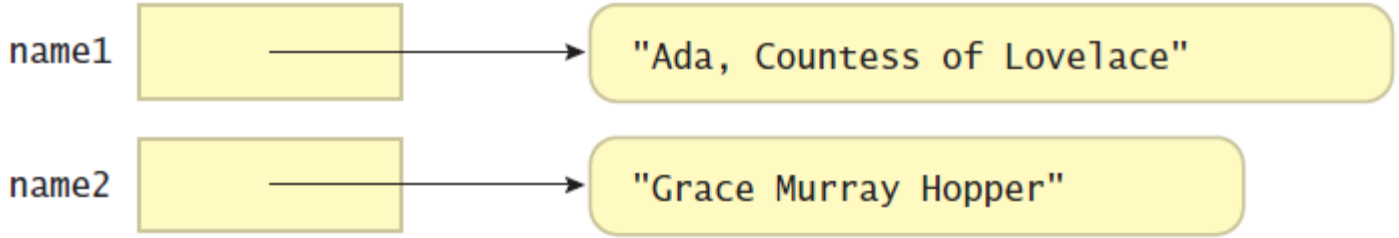
```
com.sun.javafx.geom.Rectangle[x=5,y=10,width=20,height=30]
com.sun.javafx.geom.Rectangle[x=5,y=10,width=20,height=30]

com.sun.javafx.geom.Rectangle[x=20,y=35,width=20,height=30]
com.sun.javafx.geom.Rectangle[x=20,y=35,width=20,height=30]
```

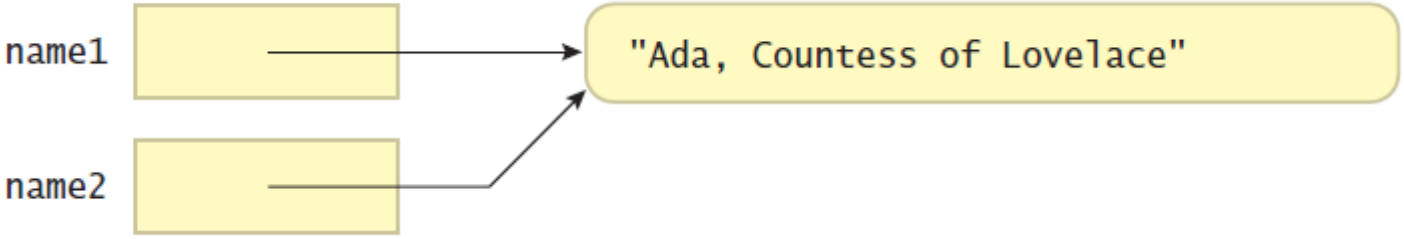
ตัวแปรอ้างอิงและการเกิดสมนาม (ต่อ)

```
4   public static void main(String[] args)
5   {
6       String name1 = "Ada, Countess of Lovelace";
7       String name2 = "Grace Murray Hopper";
8       name2 = name1;
9       System.out.println(name2);
10  }
```

6,7



8



การกำหนดตัวแปรคงที่ constant variables

- ไม่ว่าตัวแปรจะเป็นตัวแปรชนิดใดก็ตาม หากไม่ต้องการให้ตัวแปรนั้น ๆ เปลี่ยนค่าหลังจากกำหนดค่าเริ่มต้นแล้ว ให้กำหนดตัวแปรนั้นเป็น **final**
- การกำหนดค่าเริ่มต้น สามารถทำได้เหมือนกับ instance variable
- ตัวแปรของคลาสที่ไม่สามารถเปลี่ยนค่าได้ (static final) มีธรรมเนียมในการตั้งชื่อด้วย CAPITAL_LETTER

Static constants

```
public class Math {
    ...
    public static final double PI = 3.14159265358979323846;
    ...
}
```

You can access PI directly using Math.PI without having to create an instance of Math

EX: double vPi = Math.PI;

Sharing objects using static consta

```
public class Employee {
    private static final Random generator = new
Random();
    private int id;
    ...
    public Employee() {
        id = 1 + generator.nextInt(1_000_000);
    }
}
```

ค่า null

- กำหนดค่าว่างให้กับตัวแปรอ้างอิงได้เท่านั้น

```
private int first = null; //NOT OK
private String name = null; //OK
```

- การนำตัวแปรที่อ้างอิงค่าว่างไปใช้ เสมือนว่าตัวแปรนั้นอ้างอิงข้อมูลอยู่ จะทำให้เกิดข้อผิดพลาดขึ้นในรูปของสิ่งผิดปรกติ (Exception) ที่ชื่อว่า

NullPointerException

Method

Method

```
<class-modifiers> class ClassName {
    <field-declarations>
    <constructor-declarations>
    <method-declarations>
}
```

- ความสามารถ / พฤติกรรม ถือเป็นสมาชิกของคลาสอย่างหนึ่ง
- แบ่งออกเป็น 5 ชนิด

Method (ต่อ)

```
<method-modifiers> <return-type> methodName(<parameter-list>)
    <method-body>
```

<method-modifiers> เป็นตัวกำหนดคุณสมบัติของเมทอด ประกอบด้วยชนิด และการเข้าถึง

ชนิด(เลือกได้มากกว่า 1 ตัว): abstract, static, final เป็นต้น

การเข้าถึง (เลือกได้ตัวเดียว): private, protected และ public

<return-type> เป็นตัวระบุชนิดของข้อมูลที่จะส่งกลับจากเมทอด โดยที่ **void** จะหมายถึงเมทอดนี้ไม่ส่งค่ากลับ

<parameter-list> เป็นรายการพารามิเตอร์ที่เมทอดนี้จะรับ

<method-body> เป็นตัวโค้ดของเมทอดในรูปของบล็อก

Method (ต่อ)

1. **Constructor method** คือ Method ที่มีการกำหนดชื่อ Method ให้เป็นชื่อเดียวกับชื่อ Class ใช้กำหนดค่าเริ่มต้น
2. **Instance Method** คือ ต้องเรียกใช้ต้องสร้าง Object ขึ้นมาใช้งาน ถูกนำมาใช้งานบ่อยที่สุด
3. **Static Method** คือ Method ที่สามารถเรียกใช้โดยไม่ต้องสร้าง Object ขึ้นมาใช้งาน
4. **Overloading Method** คือ Method หลาย ๆ Method ที่มีชื่อเหมือนกัน แตกต่างที่ค่า Argument ที่แตกต่างกัน
5. **Overriding Method** คือ Method ที่มีลักษณะที่ Class ลูกสามารถเขียนทับ Method ของ Class แม่ได้

Constructor ตัวสร้าง

```
<constructor-modifiers> ClassName(<parameter-list>)
    <constructor-body>
```

เป็นเมทอดชนิดพิเศษที่มีหน้าที่กำหนดสถานะตั้งต้นให้กับอ็อบเจกต์ที่สร้างขึ้นใหม่แต่ละตัว มีชื่อเดียวกับคลาส ถูกเรียกใช้เป็นอันดับแรกเมื่อสร้างวัตถุ

- <constructor-modifiers> เป็นตัวกำหนดคุณสมบัติของเมทอด ซึ่งอาจจะเป็น private, protected, public หรือละไว้ได้
- <parameter-list> เป็นรายการพารามิเตอร์ที่คอนสตรักเตอร์นี้จะรับ
- <constructor-body> เป็นตัวโค้ดของคอนสตรักเตอร์ในรูปของบล็อก

Constructor/ตัวสร้าง (ต่อ)

```
public Employee(String name, double salary) {
   this.name = name;
   this.salary = salary;
}
```

CAUTION: Never specify return type for a construc

```
// This is wrong!
public void Employee(String name, double salary)
```

Constructor/ตัวสร้าง (ต่อ)

ถึงแม้ว่าเราไม่ระบุการคืนค่าใด ๆ ใน constructors แต่ในความจริงแล้วมีการคืนค่าที่อยู่ของอ็อบเจกต์ที่ถูกสร้างขึ้นให้กับตัวแปรอ้างอิง หรือ เมทอด ก็ได้

```
// We can save it to a variable
Employee james = new Employee("James Bond", 50000);
// Or we can pass it to a method
ArrayList<Employee> staff = new ArrayList<>();
staff.add(new Employee("James Bond", 50000));
```

ตัวอย่าง Constructor

```
1  import java.util.Random;
2
3  public class Employee2
4  {
5      private String name;
6      private double salary;
7      private static int lastId = 0;
8      private int id;
9
10     // Constructor Method
11     public Employee2(String name, double salary)
12     {
13         lastId++;
14         this.id = lastId;
15         this.name = name;
16         this.salary = salary;
17     }
18 }
```

ตัวอย่าง Constructor เรียกใช้ constructor

```java
// Parameter-less constructor -- default to zeros
public Pair() {
    this(0, 0);
}

// Two-parameter constructor
public Pair(int first, int second) {
    this.first = first;
    this.second = second;
}
```

ช่วยลดการซ้ำซ้อนกันของโค้ด

Default constructors

Aclass with no constructors is automatically given a default constructor with no parameters

Default constructors do nothing!

Argument and Parameter

Argument คือ ตัวแปรที่ส่งไปให้เมธอดพร้อมกับการเรียกใช้เมธอด ในกรณีมีจำนวนมากกว่าหนึ่งค่าให้คั่นด้วยเครื่องหมาย “,”

Parameter คือ ตัวแปรที่ทำหน้าที่รับค่าอาร์กิวเมนต์ที่ส่งมาใช้งานในเมธอด ในกรณีมีจำนวนมากกว่าหนึ่งค่าให้คั่นด้วยเครื่องหมาย “,”

Argument ต้องมีจำนวนเท่ากับตัวแปรที่เป็น **Parameter**

Datatype Argument กับ **Parameter** ในแต่ละตำแหน่งจะต้องสอดคล้องกัน

Argument and Parameter

Lab8CalCircle

```
7    public double calArea( int radius)
8    {
9        return (double)(Math.PI * Math.pow(radius, 2));
10   }
11
12   public static double calFerence(int radius2)
13   {
14       return (double)(2 * Math.PI * radius2);
15   }
16
17   public static void main(String[] args)
18   {
19       int radiusNew;
20       double area, circumFerence;
21       Scanner scan = new Scanner(System.in);
22
23       System.out.print("Enter the circle's radius: ");
24       radiusNew = scan.nextInt();
25
26       Lab8CalCircle lab8 = new Lab8CalCircle();
27       area = lab8.calArea(radiusNew);
28
29       //area = calArea(radius);
30       circumFerence = calFerence(radiusNew);
31
32       DecimalFormat fmt = new DecimalFormat("0.###");
33       System.out.println("The circle's area: " + fmt.format(area));
34       System.out.println("The circle's circumference: " + fmt.format(circumFerence));
35   }
```

Parameter

Argument

Java has call-by-value semantics

When calling a method, arguments are always copied onto parameter variables

This is difference from some languages which have call-by-reference semantics, in which the pointer (or reference) to an argument variable is passed instead

ตัวอ้างอิงอ็อบเจกต์ปัจจุบัน (this)

- คำสั่ง this ถูกนำมาใช้เพื่อให้สามารถอ้างอิง ตัวแปร เมทอด และคอนสตรักเตอร์ของตัวอ็อบเจกต์เองได้
- นิยมมาใช้ในกรณีถูกบดบังจากตัวแปร local

```
class Juggler {
    private int value;

    public Juggler(int value) {
        this.value = value;
    }

    public void swapWith(Juggler another) {
        int temp = this.value;
        this.value = another.value;
        another.value = temp;
    }

    public int getValue() {
        return value;
    }
}
```

Instance Method

Method ที่เรียกผ่าน Object ที่สร้างจาก Class ด้วยตัวดำเนินการ new ซึ่ง Method ที่สร้างเพื่อการใช้งานทั่ว ๆ ไป

```
4⊖    public static void main(String[] args)
5     {
6         SimpleBox box1 = new SimpleBox(1);
7         SimpleBox box2 = null;
8
9         System.out.println("Box1 : "+box1.value+"\n");
10
11        box2 = box1;
12        box1.value = 3;
13
14        System.out.println("Box1 : "+box1.value);
15        System.out.println("Box2 : "+box2.value);
16
17    }
```

การเข้าถึงและการเปลี่ยนแปลงข้อมูลของอ็อบเจกต์ (Getter/Setter)

- **Getter/Setter** คือเป็น **instance method** ที่ใช้เข้าถึงและเปลี่ยนแปลงข้อมูล(ในตัวแปร **instance**) โดยเฉพาะ
- ทำไมต้องมี? เพราะการพัฒนาโปรแกรมจริง อาจมีผู้พัฒนาร่วมกันหลายคน ซึ่งมักเกิดปัญหา
  - การเปลี่ยนแปลงค่าในรูปแบบที่เราไม่อนุญาต
  - ขั้นตอนหรือวิธีการเข้าถึงข้อมูลอาจไม่เป็นไปตามที่เรากำหนด ไม่เป็นไปตามเวลาที่คาดไว้
  - โปรแกรมส่วนอื่นไปขึ้นตรงกับตัวแปรภายในอ็อบเจกต์นี้ (เพราะไปเรียกใช้โดยตรง ไม่ยอมผ่านตัวกรอง)

การเข้าถึงและการเปลี่ยนแปลงข้อมูลของอ็อบเจกต์ (ต่อ)

- แนวทางการแก้ไขปัญหานี้โดยทั่วไป
  - กำหนดตัวแปรด้วย **private**
  - กำหนด **method getter** ให้กับตัวแปรที่อยากให้เข้าถึงแต่ละตัว
  - กำหนด **method setter** ให้กับตัวแปรที่อยากให้เปลี่ยนแปลงได้ในแต่ละตัว
- ธรรมเนียมในการตั้งชื่อ **method** ให้ขึ้นต้นด้วย **get** และ **set**
- **Getter/Setter** ไม่จำเป็นต้องทำรายตัวแปร แต่สามารถทำเป็นรายกลุ่มได้
- **Method** ที่ใช้สำหรับตรวจสอบสถานะของอ็อบเจกต์ มักส่งค่ากลับเป็นจริงหรือเท็จ ให้ตั้งชื่อ **method** ที่ขึ้นต้นด้วย **“is...”**

ตัวอย่างการสร้าง getter/setter

```java
public class Pair {
    private int first;
    private int second;

    public int getFirst() {
        return first;
    }

    public int getSecond() {
        return second;
    }

    public void setFirst(int value) {
        first = value;
    }

    public void swap() {
        int temp = first;
        first = second;
        second = temp;
    }
```

การกำหนดค่าตั้งต้นโดยใช้บล๊อก

- การเรียกใช้ method constructor ไม่ใช่วิธีการเดียวสำหรับการกำหนดข้อมูลตั้งต้นให้กับตัวแปร
- สามารถใช้สิ่งที่เรียกว่า initializer block ได้
- ลำดับความสำคัญในการทำงานคือ static variables -> static initializer block -> instance variables -> instance initializers block -> constructors
- static initializer block จะทำเพียงครั้งเดียวตอนที่คลาสถูกโหลด หากเราสร้างอ็อบเจ็กต์ใหม่อีกครั้ง ส่วนเหล่านั้นจะไม่ถูกเรียกอีก จะเหลือแต่บล็อกกำหนดค่าตั้งต้นแบบไม่สถิตและคอนสตรักเตอร์ที่จะถูกเรียก
- ถ้ามี initializer block หลายกลุ่ม ให้เรียงความสำคัญจากบนลงล่าง

Static initialization blocks

```
public class CreditCardForm {
    private static final ArrayList<Integer> expirationYear = new
ArrayList<>();
    static {
        // Add the next twenty years to the array list
        int year = LocalDate.now().getYear();
        for (int i = year; i <= year + 20; i++) {
            expirationYear.add(i);
        }
    }
    ...
}
```

Initializer example

```
E:\playground\fund2\week04>java InitializerExample
Static initializer block executed and count = 0
First initializer block executed and count = 1
Second initializer block executed and count = 1
Constructor called and count = 1
```

```java
class Initializer {
    private int v1;
    private int v2;
    private static int start = 1;
    private static int count;
    // First non-static initializer block
    {
        System.out.println("First initializer block executed and count = "+count);
        v1 = 1;
    }
    // Constructor
    public Initializer() {
        System.out.println("Constructor called and count = "+count);
        v2 = 2;
    }
    // Second non-static initializer block
    {
        System.out.println("Second initializer block executed and count = "+count);
        v1 = v1 + v2;
    }
    // Static initializer block
    static {
        System.out.println("Static initializer block executed and count = "+count);
        count = start;
    }
}
```

Overloading Method

- เป็น **Method** ที่มีคุณลักษณะที่มีได้หลายรูปแบบ (**Polymorphism**) โดยใช้ชื่อ **Method** เดียวกันมากกว่า 1 **Method** เพื่อทำงานในแบบเดียวกัน
- สิ่งที่ต่างกันคือชนิด **Datatype** ของผลลัพธ์, จำนวน, **Datatype** ของ **Argument** ที่ใช้ในการรับข้อมูลแตกต่างกัน

Overloading constructors

```
// We can have another constructor
public Employee(double salary) {
   this.name = "";
   this.salary = salary;
}
// Then these calls are valid
Employee james = new Employee("James Bond", 50000);
   // calls Employee(String, double) constructor
Employee anonymous = new Employee(40000);
   // calls Employee(double) constructor
```

Overloading Method

```
2  public class Employee3
3  {
4          private String name;
5          private double salary;
6          private static int lastId = 0;
7          private int id;
8
9          // Constructor Method
10         public Employee3()              // Overload with duplicate
11         {
12             //Employee("",0.00);     //incorrect
13             this("",0.00);
14         }
15         public Employee3(double salary)
16         {
17             this("",salary);           // Overload with duplicate
18         }
19         public Employee3(String name, double salary)
20         {
21             lastId++;
22             this.id = lastId;
23             this.name = name;
24             this.salary = salary;
25         }
26 }
```

3

2

1

Overloading Method (ต่อ)

- ชนิดข้อมูล และจำนวนพารามิเตอร์ ถูกใช้เป็นเกณฑ์ในการเลือก method เพื่อทำงาน
- หากไม่มีการสร้าง method ที่พอดีกับการเรียกใช้จะทำอย่างไร ?
  - **Type widing** เกิดขึ้นเมื่อเรียกใช้ overloading method ที่มีชนิด parameter ไม่ตรงกับ argument ตัวแปลภาษาจะเลือกใช้ **method** ที่มี **parameter** ขนาดใหญ่กว่า **argument** (มีลำดับความสำคัญสูงกว่า auto boxing)
  - Auto boxing ตัวแปลภาษาจะเลือกแปลง wrapping class <-> primitive type

Static Method

- เป็น Method ที่เรียกใช้ได้โดยไม่ต้องสร้าง Object สามารถเรียกใช้ Method ประเภทนี้ผ่านชื่อ Class ได้เลย แต่จะต้องเรียกใช้จาก Method ประเภท static method เหมือนกัน

ตัวอย่างการเรียกใช้

```
Math.pow(x, a)
Here's why:
public class Math {
    public static double pow(double base, double exponent) {
        ...
    }
}
```

ตัวอย่างการเรียกใช้ static method (ต่อ)

```
public class RandomNumbers {
    public static int nextInt(Random generator, int low, int high)  {
        return low+generator.nextInt(high - low+1);
    }
}

// Then we can use:
int dieToss = RandomNumbers.nextInt(gen, 1, 6);
```

ตัวอย่างการเรียกใช้ static method (ต่อ)

```
public class RandomNumbers {
   private static Random generator = new Random(
   public static int nextInt(int low, int high) {
      return low + generator.nextInt(high - low + 1);
         // OK to access the static generator varia
   }
}

// So we can use:
int dieToss = RandomNumbers.nextInt(1, 6);
```

Why use static methods?

Static methods and variables are also called class methods and class variables

They can be accessed directly through the class without needing to create an object

Static methods can only access static variables

They are also often "factory" methods

```
NumberFormat currencyFormatter = NumberFormat.getCurrencyInstance();
NumberFormat percentFormatter = NumberFormat.getPercentInstance();
double x = 0.1;
System.out.println(currencyFormatter.format(x)); // Prints $0.10
System.out.println(percentFormatter.format(x)); // Prints 10%
```

การเรียก method ต่อเป็นทอด (Chaining methodalls)

หากการเรียกใช้ method มีการคืนค่ากลับเป็น ตัวแปรอ้างอิง สามารถเรียกใช้ method ที่เป็นสมาชิกของตัวแปรอ้างอิงนั้นได้เลย เช่น

```java
import java.time.LocalDateTime;
import java.time.DayOfWeek;

// No method chaining
LocalDateTime now = LocalDateTime.now();
DayOfWeek day = now.getDayOfWeek();
DayOfWeek nextDay = day.plus(1);
System.out.println("1. Next day is " + nextDay);

// With method chaining
System.out.println("2. Next day is " + LocalDateTime.now().getDayOfWeek().plus(1));
```

```
1. Next day is SUNDAY
2. Next day is SUNDAY
```

Inner class
And
Static nested

คลาสซ้อน

- Java อนุญาตให้สร้างคลาสภายในคลาส และ สร้างคลาสภายในเมทอด
  - หากคลาสที่ซ้อนอยู่ไม่ใช่ static class จะเรียกคลาสที่นำมาซ้อนว่า “Inner class”
  - หากคลาสที่ซ้อนอยู่เป็น static class จะเรียกว่าคลาสซ้อนแบบสถิต (static nested class)
- คลาสภายในจะเข้าถึงสมาชิกของคลาสภายนอกได้ทั้งหมด
- ถ้าต้องเข้าถึงสมาชิกที่ถูกบดบังอยู่ให้เรียกใช้โดย OUTTERCLASSNAME.this.MEMBER
- คลาสซ้อนแบบสถิตไม่สามารถเข้าถึงสมาชิกใด ๆ ที่ไม่ใช่ static ได้

ตัวอย่าง
คลาสซ้อน

```java
public class OuterClass {
    public int publicVar = 0;
    private int privateVar = 1;

    private class InnerClass {
        public int publicVar = 2;
        private int privateVar = 3;

        public void innerClassTest() {
            System.out.println(publicVar); // 2
            System.out.println(privateVar); // 3
            System.out.println(OuterClass.this.privateVar); // 1
            System.out.println(StaticNestedClass.staticClassVar); // 4
        }
    }

    public static class StaticNestedClass {
        public static int staticClassVar = 4;
    }

    public void outerClassTest() {
        System.out.println(publicVar); // 0
        System.out.println(privateVar); // 1

        InnerClass inner = new InnerClass();
        inner.innerClassTest();
    }

    public static void main(String[] args) {
        OuterClass top = new OuterClass();

        System.out.println(top.publicVar); // 0
        System.out.println(top.privateVar); // 1

        top.outerClassTest();
    }
}
```

Exception and it's handler

Exception and Exception Handler

- ในทางปฏิบัติแล้ว โค้ดควรมี **“การตรวจสอบข้อผิดพลาด”** และ **“การจัดการข้อผิดพลาด”**
- **“การตรวจสอบข้อผิดพลาด”** คือ การสร้างกลไกสำหรับการตรวจสอบสถานะหรือค่าต่าง ๆ ที่ผู้เขียนโค้ดต้องการเฝ้าระวัง เพื่อรายงานข้อผิดพลาด และสามารถรตามรอยความผิดพลาดได้ง่ายขึ้น ปกติแล้วจะใช้กลไกของ exception ที่มาพร้อมกับภาษาที่ใช้

```
public DownCounter(int initialCount) {
    if (initialCount <= 0) {
        throw new IllegalArgumentException("Initial count must be > 0");
    }

    count = initialCount;
}
```

ตัวอย่าง Exception

1. ArrayIndexOutOfBoundException อ้างถึงสมาชิกภายในอาร์เรย์ไม่ถูกต้อง
2. ArithmeticException การดำเนินการทางคณิตศาสตร์ไม่ถูกต้อง
3. NullPointerException อ้างถึงค่าที่เป็น Null เช่น เรียก Object ที่ไม่ได้ถูกสร้าง
4. IOException เป็นข้อผิดพลาดที่ภาษา Java หากมีการเรียกใช้เมธอดที่อาจเกิดข้อผิดพลาด เช่น
   - EOFException เป็นความผิดพลาดที่เกิดจากการระบุจุดสิ้นสุดของไฟล์ไม่ถูกต้อง
   - FileNotFoundException เป็นความผิดพลาดที่เกิดจากการไม่พบไฟล์ที่ต้องการ

Exception and Exception Handler (ต่อ)

- **"การจัดการข้อผิดพลาด"** คือ กลไกที่รองรับการรายงานข้อผิดพลาดที่เกิดจากคำสั่ง throw
- ผู้เขียนโค้ดต้องตัดสินใจเองว่า หากดักจับ exception ได้แล้ว จะจัดการกับสิ่งที่ผิดพลาดนั้นอย่างไร

```
try
    body
catch (E1 x1)
    catch-body-1
catch (E2 x2)
    catch-body-2
    .
    .
    .
finally
    finally-body
```

Garbage collection

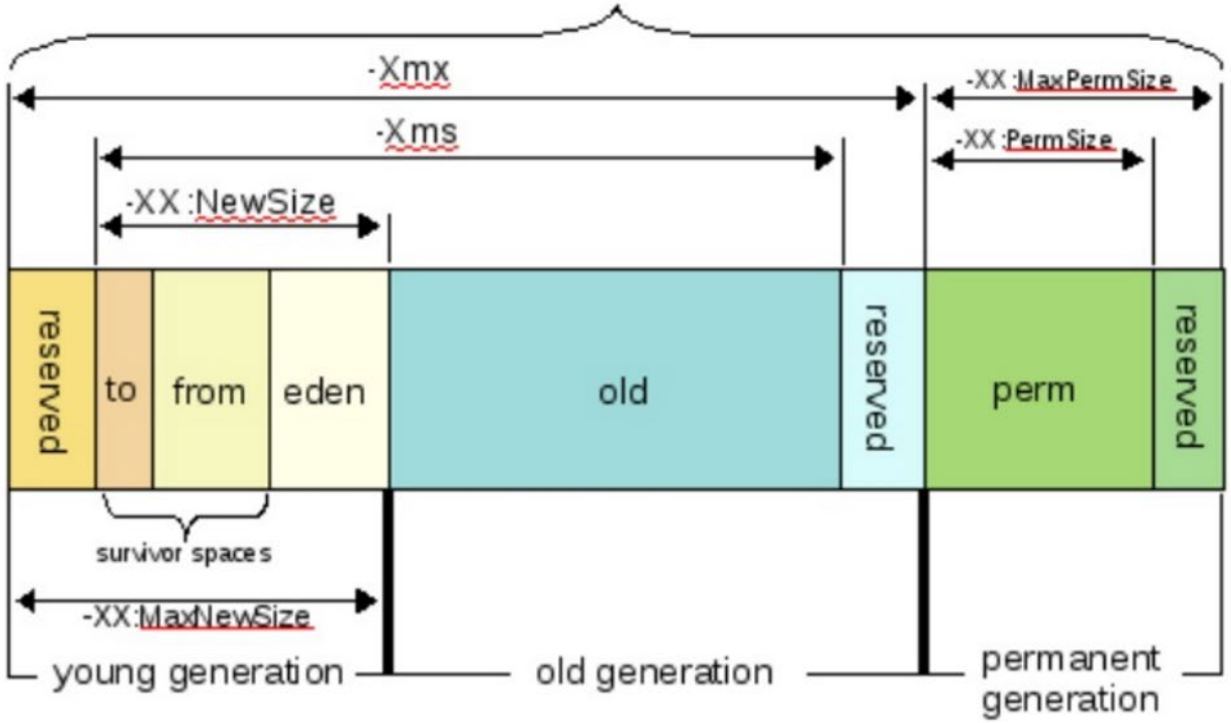
Garbage collection

```
LocalDate date = LocalDate.of(2017, 2, 6);
date = date.plusDays(1);
```

What happens to the first date?

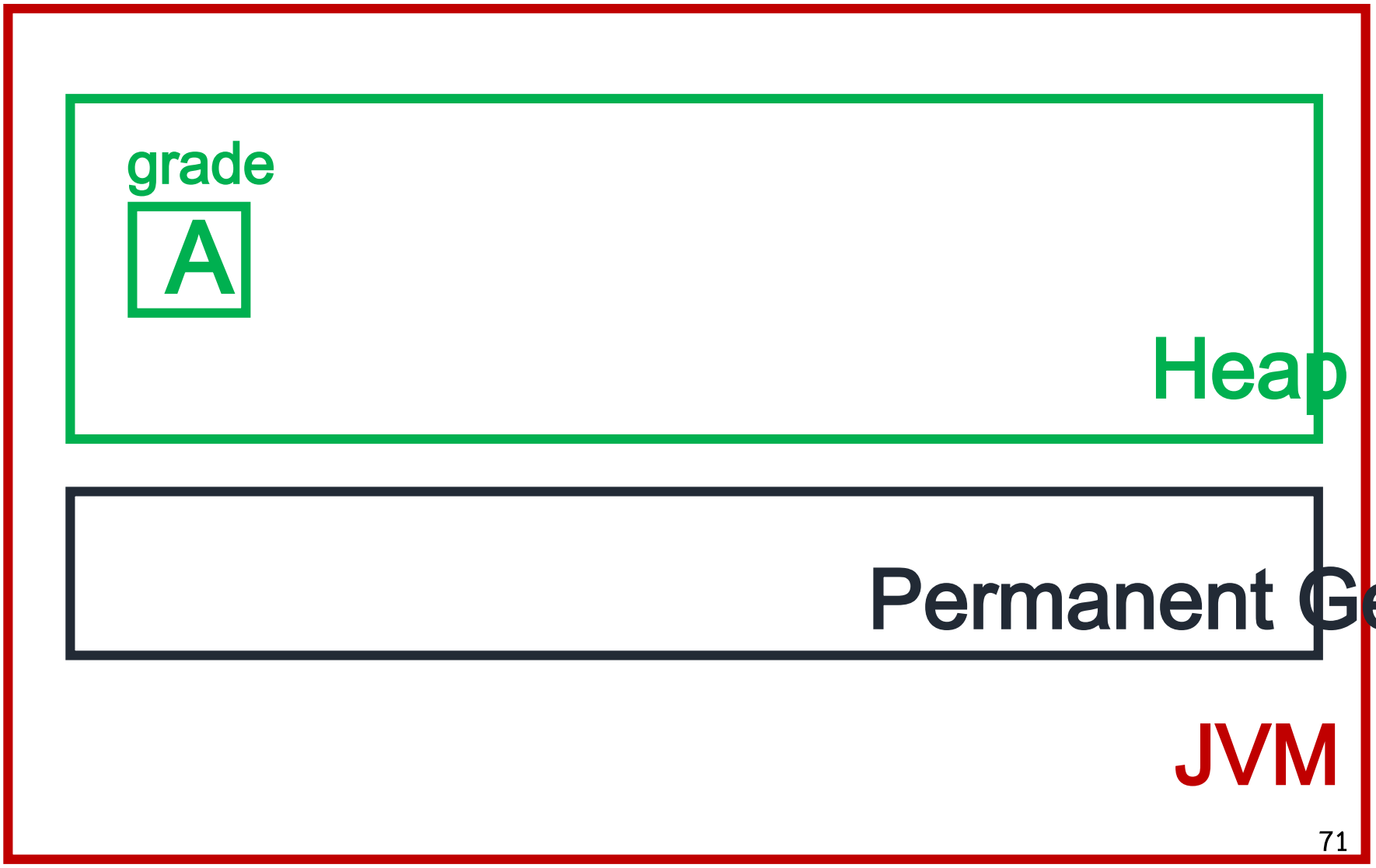
JVM

- Two heap memory types
  - Young Generation Memory
    - To/From Survivor spaces
    - Eden space
  - Old Generation Memory
- Other memory
  - Permanent Generation
  - Native memory (because of native memory references, to the underlying OS)



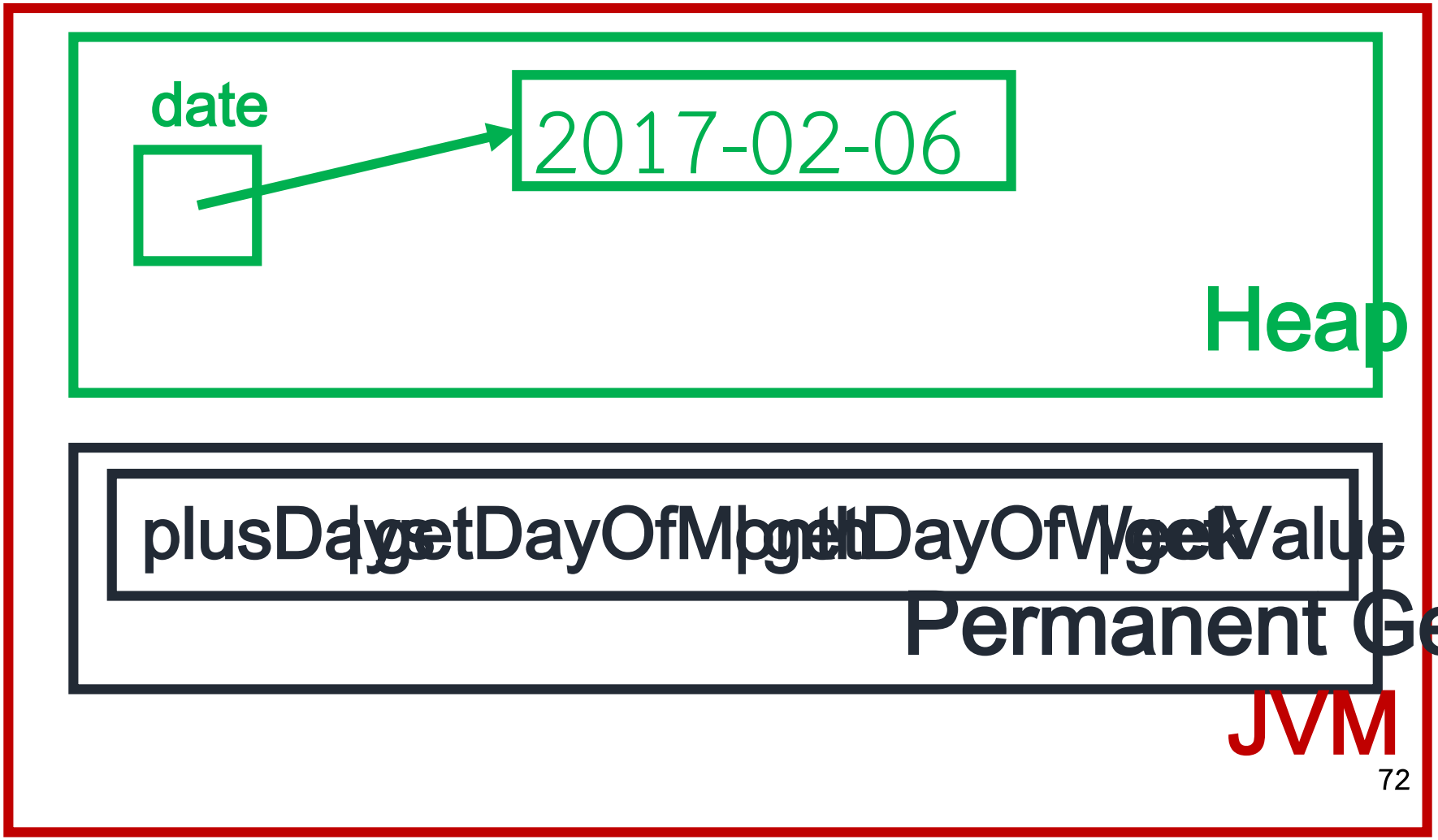
Java Memory

Ex. char grade = 'A';



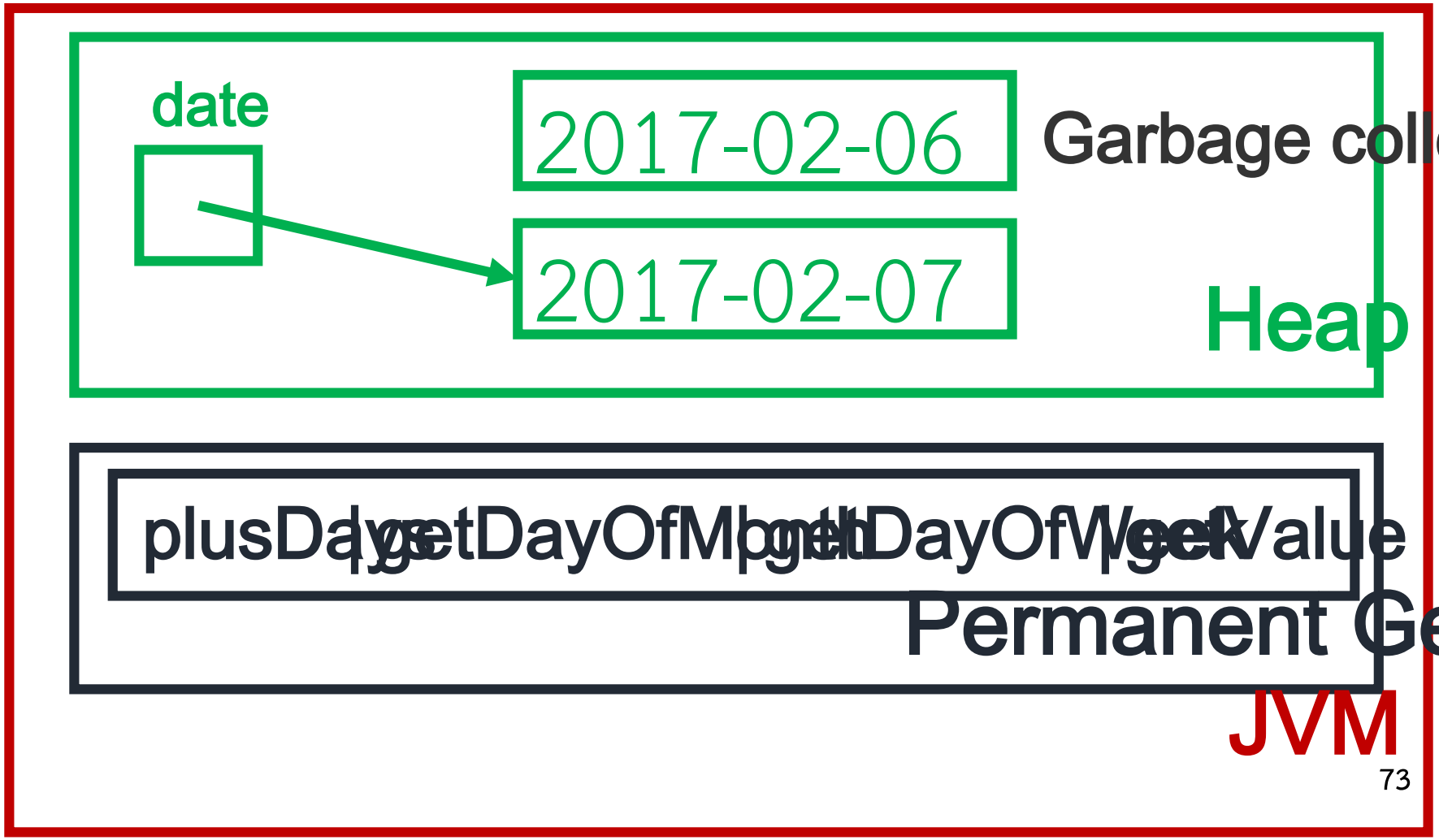
Java Memory

```
LocalDate date = LocalDate.of(2017, 2, 6);
```



Java Memory

Ex. LocalDate date = LocalDate.of(2017, 2, 6);
date = date.plusDays(1);

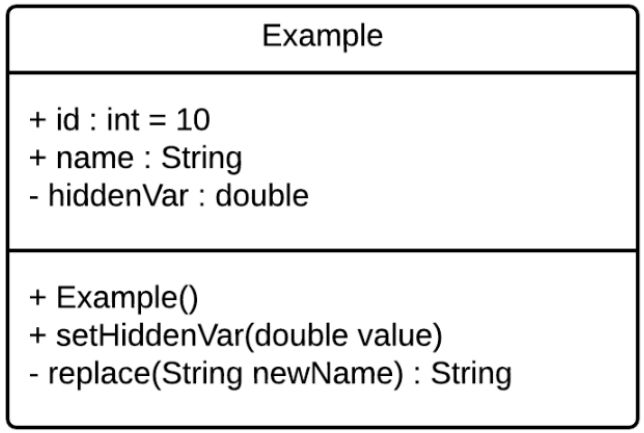


UML Class Diagram

แผนภาพคลาส

- UML (Unified Modeling Language) เป็นภาษาที่ใช้ในการแสดงโครงสร้างและการทำงานของซอฟต์แวร์ นิยมใช้เพื่อช่วยในการออกแบบและสื่อสารแบบ
- ประกอบด้วยแผนภาพหลายชนิด ขึ้นอยู่กับความเหมาะสมในการใช้งาน
- Class Diagram คือแผนภาพชนิดหนึ่งใน UML ที่นำมาใช้เพื่อสื่อสารโครงสร้างของโปรแกรม ที่พัฒนาด้วยแนวคิดเชิงวัตถุ

เปรียบเทียบแผนภาพคลาสและโค้ด




```java
public class Example {
    public int id = 10;
    public String name;
    private double hiddenVar;

    public Example() {
        // ... code omitted
    }

    public void setHiddenVar(double value) {
        // ... code omitted
    }

    private String replace(String newName) {
        // ... code omitted
    }
}
```

Summary

- Mutator methods change the state of an object — accessors don't
- Variables don't hold objects — they hold references to objects
- Instance variables and method implementations are declaration
- Instance method is invoked on an object, which is accessed via `this` reference
- Constructor has the same name as the class
- Class can have multiple (overloaded) constructors
- Static variables don't belong to any objects
- Static methods are not invoked on objects, but on classes they belong to