Week4

Programming Fundamentals II

Course Outline

1. P2J (Basic)
2. P2J (Control structures)
3. P2J (Collection types)
4. Classes and methods
5. Inheritance
6. Polymorphism
7. Interfaces
8. Testing and debugging
9. Events
10. UI programming
11. Exceptions
12. Generics
13. Concurrency
14. Team project

println / print / **printf**

println() เมื่อแสดงผลแล้วตัวชี้ตำแหน่งจะอยู่ตำแหน่งต้นของบรรทัดใหม่
print() เมื่อแสดงผลแล้วตัวชี้ตำแหน่งจะอยู่ตำแหน่งสุดท้ายของข้อมูลที่แสดงผล

printf() รูปแบบการใช้งาน

```
System.out.println(Control_String, argument1, argument2, ..., argumentn)
```

โดยที่ Control_String ประกอบด้วย รหัสควบคุมการแสดงผล, รหัสการแสดงผล และส่วนขยายรหัสการแสดงผล
argument1, argument2, argumentn เป็นข้อมูลที่ต้องการแสดงผล

println / print / **printf**

รหัสการแสดงผล	ชนิดของข้อมูลที่ใช้
%c	ตัวอักขระ
%d	เลขจำนวนเต็ม
%f	เลขทศนิยม
%e	เลขในรูป exponential
%s	ข้อความ
%u	เลขจำนวนเต็มไม่คิดเครื่องหมาย

println / print / **printf**

```
4⊖    public static void main(String[] args)
5     {
6         String  stringType  = "String";
7         int     intType     = 12345;
8         double  doubleType  = 12345.6789;
9         char    charType    = 'Y';
10
11        System.out.printf("String: %s \n",stringType);
12        System.out.printf("Integer: %d \n",intType);
13        System.out.printf("Double: %f \n",doubleType);
14        System.out.printf("Char: %c \n",charType);
15        System.out.printf("Decimal: %.2f \n",doubleType);
16
17
```

Console ⊠ Problems @ Javad

```
<terminated> Study01Printf [Java App
String: String
Integer: 12345
Double: 12345.678900
Char: Y
Decimal: 12345.68
```


Week 3 **Collections**

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

Arrays

รูปแบบ Arrays

Datatype[] ArrayName = new Datatype[n];

Ex.
```
/// Work2
char[] fund2_grade = new char[82];
```

กำหนดค่า
```
/// Work2
char[] fund2_grade = {'A', 'B', 'C', 'D'};
```

Method :	length	หาจำนวนสมาชิกของ Arrays
	sort()	เรียงลำดับข้อมูล น้อย>มาก
	binarySearch()	ค้นหาข้อมูลใน Arrays
	fill()	ใช้กำหนดค่าข้อมูลให้กับตัวแปรอาร์เรย์
	equals()	ใช้เปรียบเทียบค่าข้อมูลในตัวแปรอาร์เรย์

Arrays Search

การค้นหาใน Arrays นิยมใช้กัน 2 รูปแบบคือ

1. Sequential Search เป็นการไล่หาข้อมูลใน Arrays ตั้งแต่ index0 จนสุด

2. Binary Search เป็นการค้นหาที่ออกแบบให้สั้นลงอย่างมาก แต่การค้นหาแบบนี้ต้องทำให้ข้อมูลอยู่ในสภาพที่พร้อม คือการจัดเรียงข้อมูลใน Arrays จากน้อยไปมากเสียก่อน

```
int[] scores = { 13, 22, 27, 18, 21, 26, 11, 8, 29, 14, 15 };

Arrays.sort(scores);
System.out.println("");
for(int i = 0; i<scores.length; i++)
{
    System.out.print(scores[i] + ", ");
}
int idx = Arrays.binarySearch(scores, 22);
System.out.println("");
System.out.println("index :" +idx);
```

Console ⊠ Problems

```
<terminated> Study03ArraysMethod [Java Application] C:\Program Fi
13, 22, 27, 18, 21, 26, 11, 8, 29, 14, 15,
8, 11, 13, 14, 15, 18, 21, 22, 26, 27, 29,
index :7
```

MutiArrays

รูปแบบ MutiArrays

```
Datatype[][]  ArrayName = new Datatype[m][n];
```

Ex. Arrays[3x2]

```
int[][] mutiArry  = { {1,2}, {3,4}, {5,6} };
```

$$mutiArry = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Strings

Method :

length()	ใช้นับจำนวนตัวอักษร(Char) ใน Strings
equals()	ใช้เปรียบเทียบค่าข้อมูลชนิด String 2 ค่า
charAt()	ใช้ค้นหาและแสดงค่าตัวอักษร ที่ตำแหน่งที่ต้องการ
ValeOf()	ใช้ในการเปลี่ยนตัวแปรเป็น Strings
concat()	ใช้สำหรับเชื่อมต่อข้อมูลชนิด String
compareTo()	ใช้เปรียบเทียบค่าข้อมูลชนิด String 2 ค่า
substring()	ใช้สำหรับตัดข้อมูลชนิด String ให้เป็นข้อความย่อย

ArrayLists

รูปแบบ ArrayLists

```
ArrayList<Integer> my_grad1 = new ArrayList<Integer>();
ArrayList my_grad2 = new ArrayList();
```

Method :

add()	เพิ่มสมาชิกใน ArrayList
get()	หาค่าสมาชิกใน ArrayList ณ ตำแหน่งที่ต้องการ
remove()	สำหรับลบสมาชิก ณ ตำแหน่งที่ต้องการออกจาก ArrayList
set()	สำหรับแก้ไขค่าสมาชิก ณ ตำแหน่งที่ต้องใน ArrayList

Collections

Looping over a collection

```
Console
<terminated> Study08ForEachLoop [Java Application] C:\Program F
Scores: 13.0 is percent: 43.333333333333336
Scores: 22.0 is percent: 73.33333333333333
Scores: 27.0 is percent: 90.0
Scores: 18.0 is percent: 60.0
Scores: 21.0 is percent: 70.0
Scores: 26.0 is percent: 86.66666666666667
Scores: 11.0 is percent: 36.666666666666664
Scores: 8.0 is percent: 26.666666666666668
Scores: 29.0 is percent: 96.66666666666667
```

```
// Work 2
double[] scores_midterm = { 13.0, 22.0, 27.0, 18.0, 21.0, 26.0, 11.0, 8.0, 29.0 };

for(double calculate : scores_midterm)
{
    System.out.println("Scores: " + calculate + " is percent: " + (calculate/30*100));
}
```

enum

```
char ch = grade_A + grade_F;
System.out.println(ch);
```

HashMaps

```
Console  Problems
<terminated> Study10HashMap [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (27 ม.ค. 2560 22:29:56)
Tuesday Day of Week: 2
{Monday=1, Thursday=4, Friday=5, Sunday=7, Wednesday=3, Tuesday=2, Saturday=6}
```

Classes and methods

Programming Fundamentals II

Week 4 **Classes and Objects**

1. Class and Objects
2. Working with objects
3. Accessor and mutator methods
4. Exception
5. Garbage collection
6. Call-by-value / Call-by-reference
7. Method
8. Static variables and methods

Objects

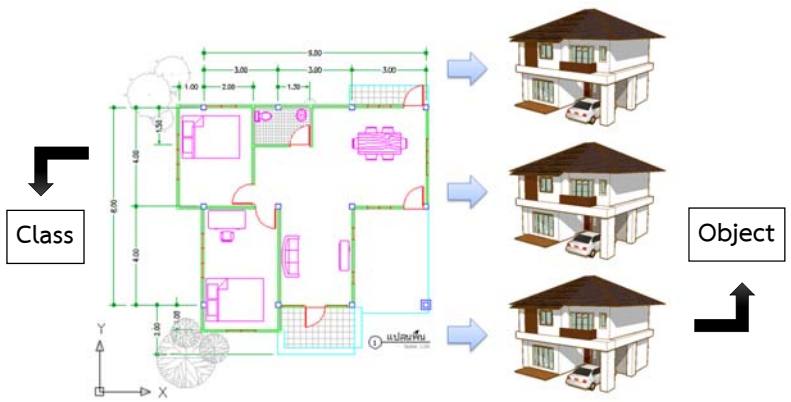
An object has a state and behaviors

A program consists of many collaborating objects

Objects

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

Class and Objects



Class and Objects

Class สิ่งที่ใช้อธิบายลักษณะและความสามารถของ Object เปรียบได้กับแม่แบบ ของ Object

Object สิ่งต่าง ๆ รอบตัว ซึ่งมีคุณลักษณะ (Attribute) และความสามารถในการทำงาน (Method)

ตัวอย่าง Object เช่น คน, รถยนต์, เครื่องคอมพิวเตอร์ เป็นต้น

Argument and Parameter

Argument ตัวแปรที่ส่งไปให้เมธอดพร้อมกับการเรียกใช้เมธอด ในกรณีมีจำนวนมากกว่าหนึ่งค่าให้คั่นด้วยเครื่องหมาย “,”

Parameter ตัวแปรที่ทำหน้าที่รับค่าอาร์กิวเมนต์ที่ส่งมาใช้งานในเมธอด ในกรณีมีจำนวนมากกว่าหนึ่งค่าให้คั่นด้วยเครื่องหมาย “,”

Argument ต้องมีจำนวนเท่ากับตัวแปรที่เป็น **Parameter**

Datatype ของ **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

Working with objects

```
LocalDate date = LocalDate.of(year, month, 1);
date = date.plusDays(1);
while (date.getMonthValue() == month) {
   System.out.printf("%4d", date.getDayOfMonth());
   date = date.plusDays(1);
   ...
}
DayOfWeek weekday = date.getDayOfWeek();
int value = weekday.getValue();
for (int i = 1; i < value; i++)
   System.out.print("    ");
```


Chaining method calls

```
int value = date.getDayOfWeek().getValue();
```


Accessor and mutator methods

Mutator changes the object on which it was invoked

Ex. rectangle.translate(10, 20); translate (); เป็น Accessor

Accessor leaves the object unchanged

Ex. Array.length(); Methode length(); เป็น Mutator

Immutable objects do not have mutator methods

Example: LocalDate is immutable

Accessor and mutator methods

```
public static void main(String[] args)
{
    // Constructor Method
    String str = new String("Programming");
    System.out.printf("%s \n",str);

    // Accessor Method
    int num = str.length();
    System.out.printf("Length : %d \n",num);

    // Mutator Methods
    str = str.concat(" Fundamental2");
    System.out.printf("%s \n",str);

}
```

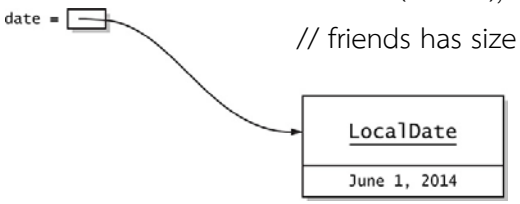
Console Problems Javadoc Decla

<terminated> Study02Mutator [Java Application] C

```
Programming
Length : 11
Programming Fundamental2
```

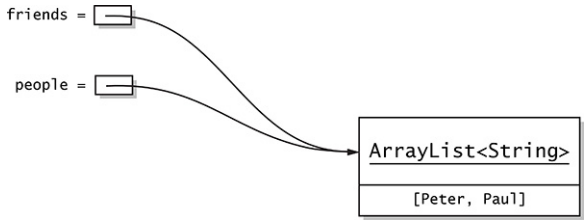
Object references

```
ArrayList<String> friends = new ArrayList<>();
    // friends is empty
friends.add("Peter");
    // friends has size 1
```



Assigning references

```
ArrayList<String> people = friends;
    // Now people and friends refer to the same object
people.add("Paul");
```



Assigning references

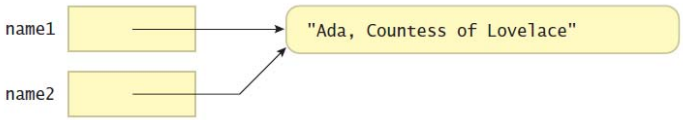
```
Rectangle box = new Rectangle(5, 10, 20, 30);      // ----------(1)
Rectangle box2 = box;                              // ----------(2)
box2.translate(15, 25);                            // ----------(3)
```

Assigning references

```
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



Null value

LocalDate date = null; // Now date doesn't refer to any object

Null values can be dangerous when they are not expected

Invoking a method on null causes a NullPointerException

Exception

RuntimeException เป็นข้อผิดพลาดที่หลีกเลี่ยงได้ ต้องเขียนให้ถูกต้อง

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

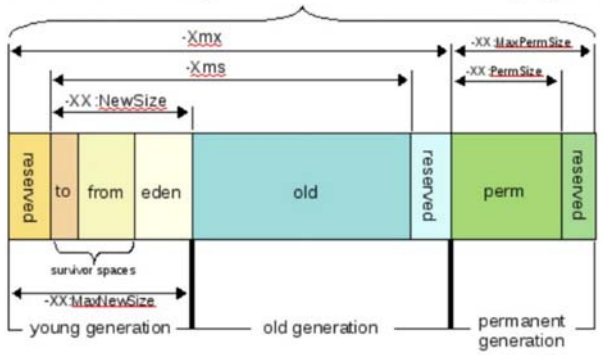
Garbage collection

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)

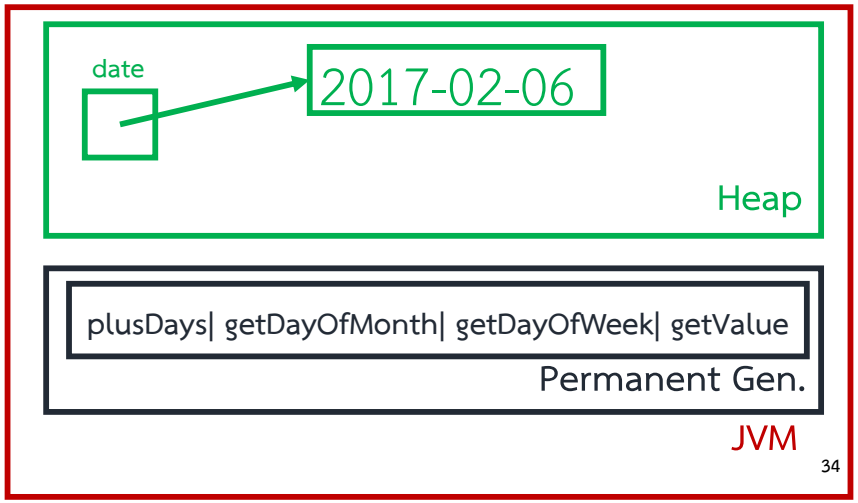


Java Memory

Ex. char grade = 'A';

Java Memory

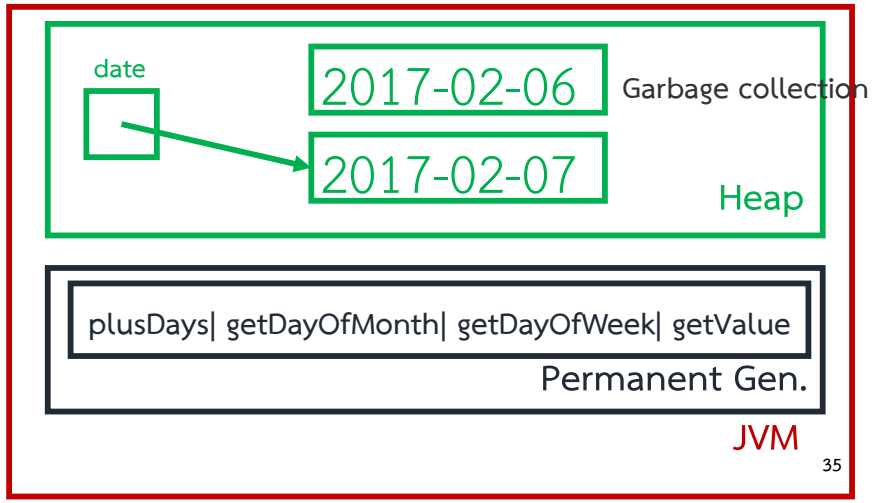
Ex. LocalDate date = LocalDate.of(2017, 2, 6);



Java Memory

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

date = date.plusDays(1);



Implementing classes

An object has its state and behaviors

State is implemented as **instance variables**

Behaviors are **implemented as methods**

A class may have their own state and behaviors which are shared among its objects

Those are called class variables and class methods (denoted by the keyword static)

Instance variables

```
public class Employee {
    private String name;
    private double salary;
    ...
}
```

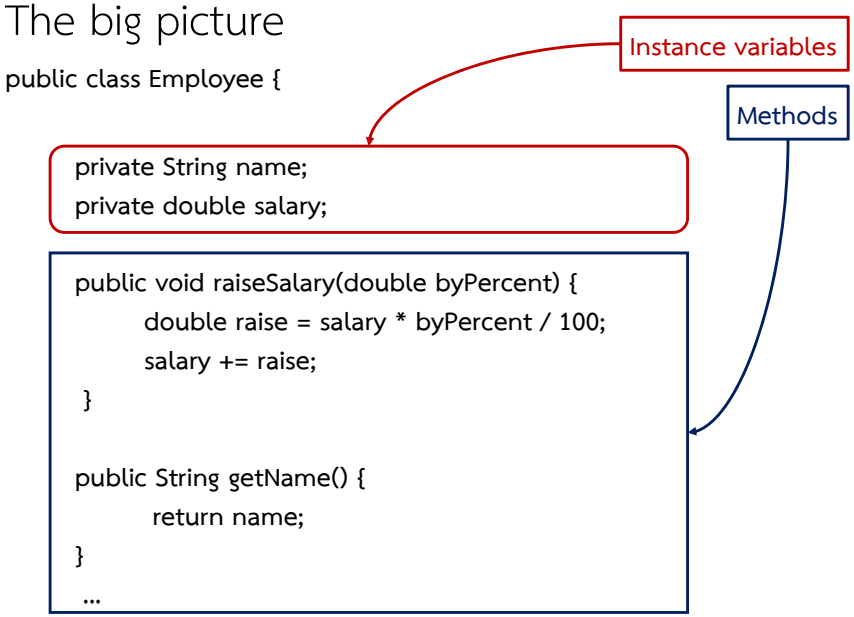
Instance variables are usually private

- To control access
- To avoid implementation dependency

Methods

```
public void raiseSalary(double byPercent) {
    double raise = salary * byPercent / 100;
    salary += raise;
}
public String getName() {
    return name;
}
```


The big picture



What happens when a method is invoked

```
// Invoking this method on fred object:
fred.raiseSalary(5);

// Cause this to happen:
double byPercent = 5;
double raise = fred.salary * byPercent / 100;
fred.salary += raise;
```

The this reference

"this" is an implicit method parameter

It is always passed into the method


```
public void raiseSalary(double byPercent) {
    double raise = this.salary * byPercent / 100;
    this.salary += raise;
}
```


Use this to avoid name clashes

```
public void setSalary(double salary) {
    this.salary = salary;
}
```


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

This won't work

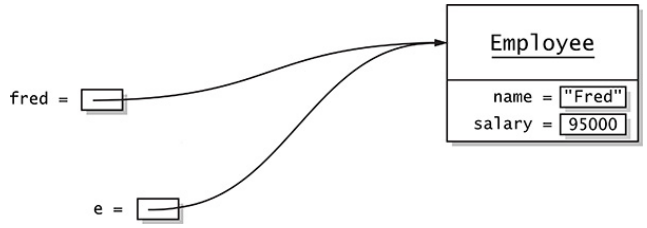
```
// If we have a method that accepts a double parameter:
public void increaseRandomly(double x) { // Won't work
    double amount = x * generator.nextDouble();
    x += amount;
}

// Then:
boss.increaseRandomly(sales);
```

Call-by-value and reference types

```
// Consider this code:
public class EvilManager {
    private Random generator;
    ...
    public void giveRandomRaise(Employee e) {
        double percentage = 10 * generator.nextGaussian();
        e.raiseSalary(percentage);
    }
}
```


Fred and e refer to the same object

```
boss.giveRandomRaise(fred);
```



The reference is copied – but this is different from call-by-reference semantics!

This doesn't work either

```
// This:
public class EvilManager {
    ...
    public void replaceWithZombie(Employee e) {
        e = new Employee("", 0);
    }
}

// Then this:
boss.replaceWithZombie(fred);
```


Call-by-value / Call-by-reference

```
4⊖    public static void main(String[] args)
5     {
6         int temp1;
7         int temp2;
8
9         temp1 = 1;
10        temp2 = 2;
11
12        System.out.println("Temp1 : "+ temp1);
13        System.out.println("Temp2 : "+ temp2);
14    }
```

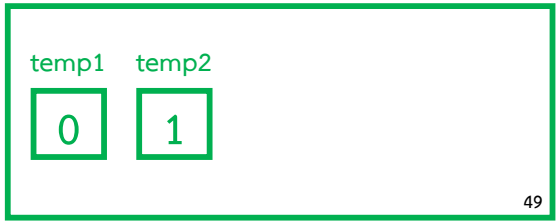
```
Console ⊠  Proble
<terminated> Study01 [J
Temp1 : 1
Temp2 : 2
```

Call-by-value / Call-by-reference

```
4    public static void main(String[] args)
5    {
6        int temp1;
7        int temp2;
8
9        temp1 = 1;
10       temp2 = temp1;
11       temp1 = 0;
12
13       System.out.println("Temp1 : "+ temp1);
14       System.out.println("Temp2 : "+ temp2);
15   }
```

Console

```
<terminated> Study02 [J
Temp1 : 0
Temp2 : 1
```

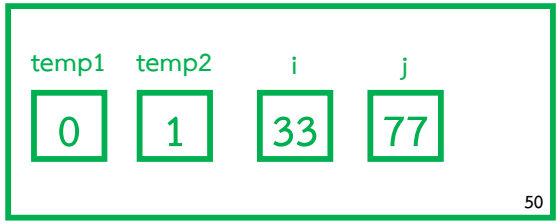


Call-by-value / Call-by-reference

```
4    public static void main(String[] args)
5    {
6        int temp1 = 1;
7        int temp2 = 2;
8        simpleMethod(temp1, temp2);
9        System.out.println("Temp1 : "+ temp1);
10       System.out.println("Temp2 : "+ temp2);
11   }
12
13   public static void simpleMethod(int i, int j)
14   {
15       i = 33;
16       j = 77;
17   }
```

Console

```
<terminated> Study02
Temp1 : 0
Temp2 : 1
```



Call-by-value / Call-by-reference

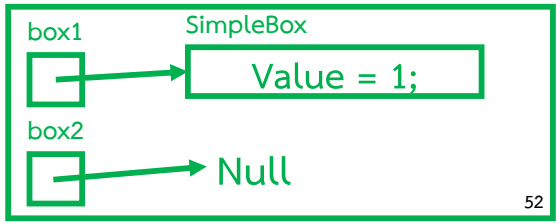
```
2  public class SimpleBox
3  {
4      // Argument
5      public int value;
6
7      // Method
8      // Constructor Method
9      public SimpleBox()
10     {
11         this.value = 0;
12     }
13     public SimpleBox(int value)
14     {
15         this.value = value;
16     }
17 }
```


Call-by-value / Call-by-reference

```
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   }
```

Console

```
<terminated> Study04
Box1 : 1
```

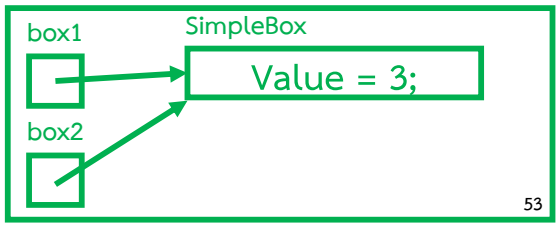


Call-by-value / Call-by-reference

```
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  }
```

Console

```
<terminated> Study04
Box1 : 1

Box1 : 3
Box2 : 3
```

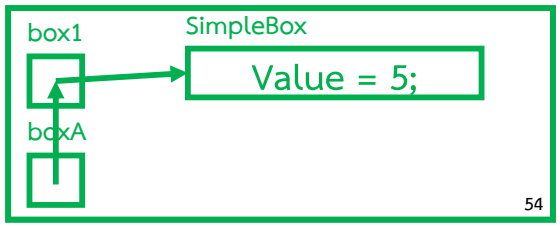


Call-by-value / Call-by-reference

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

Console

```
<terminated> Study0
Box1 : 5
```



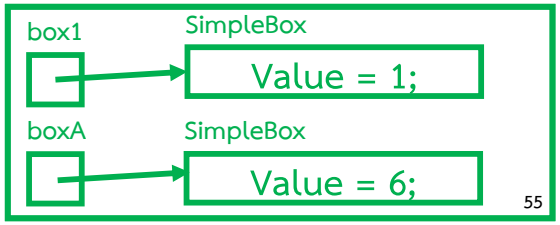
Call-by-value / Call-by-reference

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

Console

```
<terminated> Stud
Box1 : 1
```

box1 ≠ boxA



Object construction

We use new operator to construct an object

Classes can have methods that are responsible for initializing newly created objects

These methods are called **constructor method**

Constructors have the same name as the containing class

Constructors don't have return type (they don't return values)

Method

Method

Java มีรูปแบบการสร้าง Method หลายประเภท ซึ่งแต่ละประเภทมีหน้าที่ที่แตกต่างกัน แบ่งได้ดังนี้

1. Instance Method
2. Constructor method
3. Static Method
4. Overloading Method
5. Overriding Method

Method

1. Instance Method ใช้งานบ่อยที่สุด การเรียกใช้ต้องสร้าง Object ขึ้นมาใช้งาน

2. Constructor method เป็น Method ที่มีการกำหนดชื่อ Method ให้เป็นชื่อเดียวกับชื่อ Class เพื่อกำหนดค่าเริ่มต้น

3. Static Method เป็น Method ที่สามารถเรียกใช้โดยไม่ต้องสร้าง Object ขึ้นมาใช้งาน

4. Overloading Method เป็น Method หลายๆ Method ที่มีชื่อเหมือนกัน แตกต่างที่ค่า Argument ที่แตกต่างกัน

5. Overriding Method เป็น Method ที่มีลักษณะที่ Class ลูกสามารถเขียนทับ Method ของ Class แม่ได้

Instance Method

Method ที่เรียกผ่าน Object ที่สร้างจาก Class ด้วยตัวดำเนินการ new ซึ่ง Method ที่สร้างเพื่อการใช้งานทั่วๆไปเป็นเมธอดประเภท Instance method

```
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   }
```

Constructor Method

เป็น Method ที่มีการกำหนดชื่อ Method เป็นชื่อเดียวกันกับชื่อ Class เพื่อกำหนดให้ Method

1. ทำงานเป็น Method แรกเมื่อเรียกใช้งาน Class ขึ้นมา
2. การทำงานของ Constructor method จะเหมาะสำหรับการกำหนดค่าเริ่มต้นให้กับ Object

Constructors

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

CAUTION: Never specify return type for a constructor!

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


Constructor Method

```
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 }
```


Using constructors

The new operator invokes a constructor and returns a reference to the newly constructed object

```
// We can save it to a variable
Employee james = new Employee("James Bond", 500000);
// Or we can pass it to a method
ArrayList<Employee> staff = new ArrayList<>();
staff.add(new Employee("James Bond", 500000));
```

Overloading Method

- เป็น Method ที่มีคุณลักษณะที่มีได้หลายรูปแบบ (Polymorphism) โดยใช้ชื่อ Method เดียวกันมากกว่า 1 Method เพื่อทำงานในแบบเดียวกัน
- สิ่งที่ต่างกันคือชนิด Datatype ของผลลัพธ์ หรือมีจำนวนและ 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", 500000);
    // calls Employee(String, double) constructor
Employee anonymous = new Employee(40000);
    // calls Employee(double) constructor
```


Constructor may call each other

```
// This saves some duplicate code
public Employee(double salary) {
    this("", salary); // Calls Employee(String, double)
    // Other statements can follow
}
```


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 }
```

Instance variable initialization

Instance variables are initialized in one of the four ways:

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

Final instance variables

```
public class Employee {
    private final String name;
    ...
}
```

Final variables **can't change** their values once they are initialized

They may be initialized in-place or inside constructors

Initialization must be done by the end of every constructor

Default constructors

A class with no constructors is automatically given a default constructor with no parameters

Default constructors do nothing!

Static variables

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

Every instance of the Employee class share lastId

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

Sharing objects using static constants

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


One familiar face

```
public class System {
    public static final PrintStream out;
    ...
}
```


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);
        }
    }
    ...
}
```

Static Method

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

Static methods

Why can we use methods from the class Math without creating an object from Math?

Like, Math.pow(x, a)

Here's why:

```
public class Math {
    public static double pow(double base, double exponent) {
        ...
    }
}
```


Similarly

```
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);
```


Or better yet

```
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 variable
    }
}

// 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 used as “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%
```


Summary

Mutator methods change the state of an objects – accessor methods don’t

Variables don’t hold objects – they hold references to objects

Instance variables and method implementations are declared inside class declaration

Instance method is invoked on an object, which is accessible through `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