Polymorphism

การมีหลายรูป

Compiled by Kanjana Eiamsaard, 2022 -01-13

Agenda

- Polymorphism
- Adapt inheritance to polymorphism
  - Abstract class and Abstract method
  - Final class and Final method
- Composition (has-a relationship)
- Interface class
  - Java standard interface

การมีหลายรูปแบบ (Polymorphism)

- คุณสมบัติที่ทำให้เราใช้อ็อบเจกต์ที่ต่างคลาสกัน ในรูปแบบเดียวกันได้ แบ่งเป็น 4 ชนิด
  - Ad hoc polymorphism คือ การที่เมทอดชื่อเดียวกัน ทำงานต่างกัน ขึ้นอยู่กับอาร์กิวเมนต์ที่ส่งเข้ามา (Overloading)
  - Parametric polymorphism คือ การที่โค้ดไม่ได้ระบุชนิดของข้อมูลหรืออ็อบเจกต์ที่ใช้งานอย่างเฉพาะเจาะจง พบได้ในรูปของเจเนอริกส์ (generics)
  - **Subtyping คือ การใช้ตัวแปรเดียวกันอ้างอิงอ็อบเจกต์ที่ต่างคลาสกันได้**
  - Duck typing คือ การใช้งานอ็อบเจกต์โดยไม่ต้องสนใจชนิด ตราบเท่าที่อ็อบเจกต์นั้นมีเมทอดที่ต้องการ "ถ้ามันเดินเหมือนเป็ดและร้องเหมือนเป็ด มันจะต้องเป็นเป็ด"

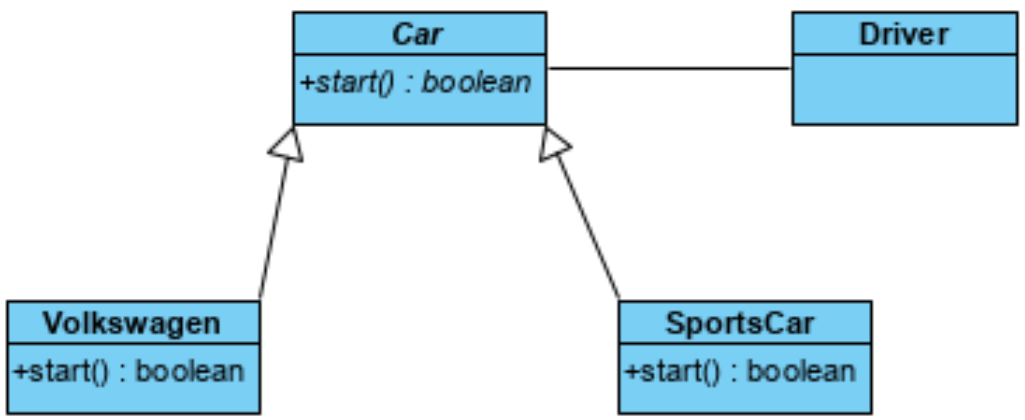
การมีหลายรูปแบบ (Polymorphism) (ต่อ)

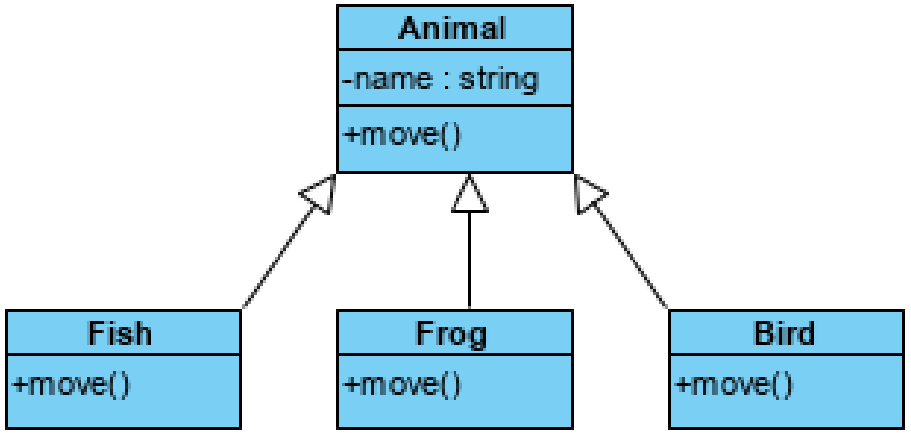
- พิจารณาเฉพาะ Subtyping คือการที่เราสามารถส่งเมสเสจ (เรียกเมทอด) ไปยังอ็อบเจกต์ได้โดยไม่จำเป็นต้องรู้ชนิดที่แท้จริงของอ็อบเจกต์นั้น
  - ข้อแม้คือ อ็อบเจกต์นั้นต้องมีเมทอดตามที่ผู้เรียกต้องการ (ทำสัญญาตกลงกัน)
  - สัญญาเกิดขึ้นเมื่อคลาสอยู่ในลำดับชั้นเดียวกัน **เกิดได้จากสายสัมพันธ์แบบ Inheritance หรือ สายสัมพันธ์แบบ Interface ก็ได้**

Polymorphism using inheritance

- Abstract class
- Abstract method
- Final class and Final method

การมีหลายรูปแบบด้วยการสืบทอด





Subclass is a Superclass BUT Superclass is not a Subclass

Fish is an Animal BUT Animal is not a Fish

Animal class example

Animal.java

week06 > Animal.java > Animal

```java
public class Animal {
    private String name;
    public Animal(String name){
        this.name = name;
    }
    public String getName(){
        return this.name;
    }
    public void move(){
        System.out.println(getName()+
        " How should I move?");
    }
}
```

Frog.java

week06 > Frog.java > Fish

```java
public class Frog extends Animal{
    public Frog(String name){
        super(name);
    }
    public void move(){
        System.out.println(this.getName()+
        " Jump Jump Jump");
    }
}
class Fish extends Animal{
    public Fish(String name){
        super(name);
    }
    public void move(){
        System.out.println(getName()+
        " is swimming");
    }
}
```

Zoo.java

week06 > Zoo.java > Zoo > main(String[])

```java
public class Zoo{
    public static void main(String[] args) {
        Animal noone = new Animal("Noone");
        Frog kero = new Frog("Kero");
        Animal anyone = new Fish("Ponyo");
        noone.move();
        kero.move();
        anyone.move();
    }
}
```

PROBLEMS 57 OUTPUT DEBUG CONSOLE TERMINAL 1: cmd

```
E:\playground\fund2\week06>java Zoo
Noone How should I move?
Kero Jump Jump Jump
Ponyo is swimming
```

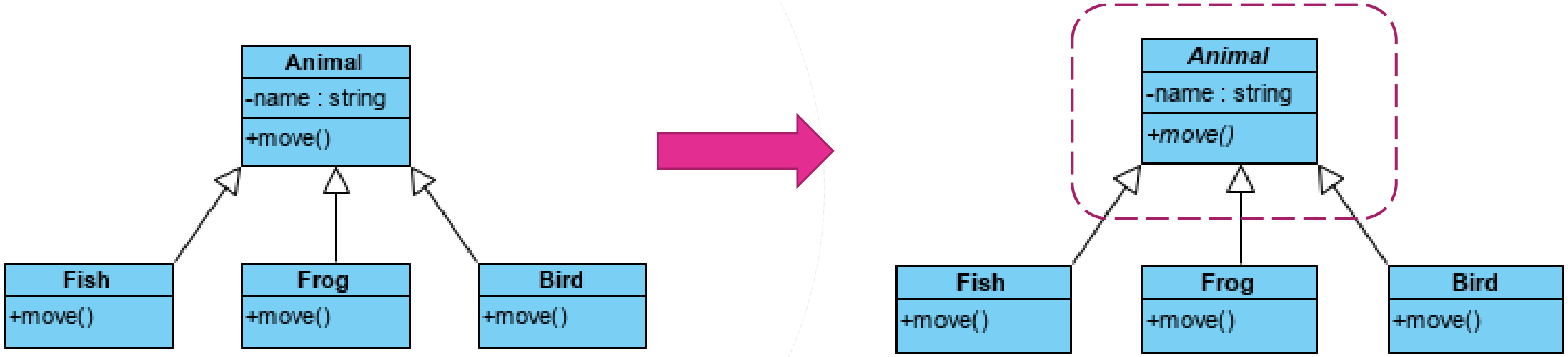
การมีหลายรูปแบบด้วยการสืบทอด (ต่อ)

- ตัวแปร noone และ anyone แสดงให้เห็นว่า superclass Animal ทำให้เกิดการมีหลายรูป กล่าวคือ ตัวแปรชนิดดังกล่าวสามารถเป็นตัวแปรอ้างอิงให้กับอ็อบเจกต์ต่างชนิดกันได้
- **แต่!!!!! Animal ไม่ควรมีตัวตนเกิดขึ้น**

การมีหลายรูปแบบด้วยการสืบทอดจากคลาสนามธรรม (Abstract class)

- การสร้าง Abstract class เป็นวิธีที่ทำให้คลาสไม่สามารถนำไปสร้างเป็นอ็อบเจกต์ได้ (Animal ไม่ควรมีตัวตนเกิดขึ้น)
- Abstract class คือ คลาสที่แสดงถึงคุณลักษณะและพฤติกรรมร่วมของกลุ่มคลาสหนึ่ง ๆ ซึ่งพฤติกรรมในคลาสชนิดนี้ไม่จำเป็นต้องระบุรายละเอียดของพฤติกรรม หากแต่สามารถกำหนดเพียงชื่อพฤติกรรมเพียงอย่างเดียวได้ (Abstract method)
  - การสร้าง abstract method เปรียบเสมือนการให้สัญญาว่า Subclass จะมีพฤติกรรมที่ Superclass กำหนด
  - หากสืบทอด abstract method ต้องโอเวอร์ไรด์ให้หมด
  - สามารถสร้าง Abstract class ได้โดยไม่ต้องมี abstract method
- ถึงแม้ว่า abstract class ไม่สามารถนำมาสร้างอ็อบเจกต์ได้ แต่สามารถนำมาใช้เป็นชนิดตัวแปรได้ (เพื่อการอ้างอิงไปยัง subclass)

คลาสนามธรรม (Abstract class)



คลาสนามธรรม (Abstract class)

Animal.java

week06 > Animal.java > Animal > move()

```java
public abstract class Animal {
    private String name;
    public Animal(String name){
        this.name = name;
    }
    public String getName(){
        return this.name;
    }
    abstract public void move();
}
```

Zoo.java

week06 > Zoo.java > Zoo > main(String[])

```java
public class Zoo{
    public static void main(String[] args) {
        Animal noone = new Animal("Noone");
        Frog kero = new Frog("Kero");
        Animal anyone = new Fish("Ponyo");
        noone.move();
        kero.move();
        anyone.move();
    }
}
```

PROBLEMS 58 OUTPUT DEBUG CONSOLE TERMINAL

```
E:\playground\fund2\week06>javac Zoo.java
Zoo.java:3: error: Animal is abstract; cannot be instantiated
        Animal noone = new Animal("Noone");
                       ^
1 error
```

คลาสนามธรรม (Abstract class) (ต่อ)

Animal.java

week06 > Animal.java > Animal

```java
public abstract class Animal {
    private String name;
    public Animal(String name){
        this.name = name;
    }
    public String getName(){
        return this.name;
    }
    public abstract void move();
}
```

Frog.java

week06 > Frog.java > Fish

```java
public class Frog extends Animal{
    public Frog(String name){
        super(name);
    }
    public void move(){
        System.out.println(this.getName()+
        " Jump Jump Jump");
    }
}
class Fish extends Animal{
    public Fish(String name){
        super(name);
    }
    public void move(){
        System.out.println(getName()+
        " is swimming");
    }
}
```

คลาสนามธรรม (Abstract class) (ต่อ)

- บรรทัดที่ 3 Class Fish override method move จึงทำให้คลาส Fish นำไปสร้างอ็อบเจกต์ได้
- บรรทัดที่ 5 Abstract class สามารถนำมาใช้เป็นชนิดตัวแปรได้
- การเลือกเมทอดขณะรันโปรแกรม (แทนการเลือกเมทอดในขณะ compile)

เรียกว่า “การจ่ายงานแบบพลวัต (dynamic dispatch)”

```
1  public class Zoo{
2      public static void main(String[] args) {
3          Fish nemo = new Fish("Nemo");
4          nemo.move();
5          Animal anyone = new Frog("Kero");
6          anyone.move();
7          anyone = nemo;
8          anyone.move();
9      }
10 }
```

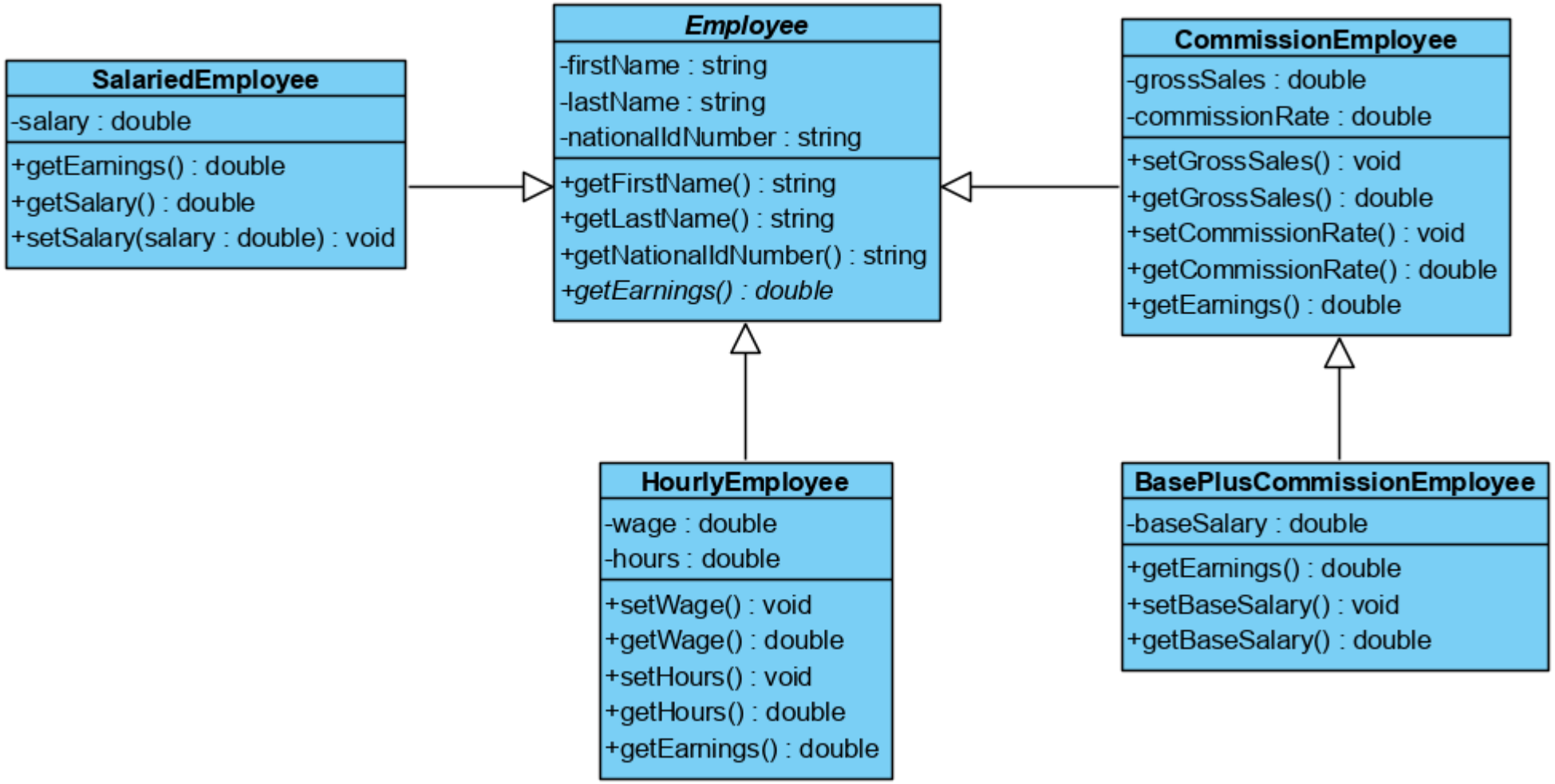
```
E:\playground\fund2\week06>java Zoo
Nemo is swimming
Kero Jump Jump Jump
Nemo is swimming
```

Payroll case study

ตัวอย่างต่อไปนี้เป็นการใช้คุณสมบัติการมีหลายรูปแบบเพื่อสร้างกลุ่มคลาสสำหรับแทนพนักงานแต่ละประเภท ได้แก่ พนักงานเงินเดือน (SalariedEmployee) พนักงานจ้างรายชั่วโมง (HourlyEmployee) พนักงานคอมมิชชัน (CommissionEmployee) และพนักงานเงินเดือนและคอมมิชชัน (BasePlusCommissionEmployee)

โดยทั้งหมดนี้สืบทอด (ทั้งโดยตรงและโดยอ้อม) จากคลาส Employee ซึ่งเป็นคลาสนามธรรม

Payroll class diagram



Focused in PayrollSystemTest

25. Polymorphism

28.Check real obj

31. Downcasting

39. What is real class of object ?

```
18          Employee[] employees = new Employee[4];
19
20          employees[0] = salariedEmployee;
21          employees[1] = hourlyEmployee;
22          employees[2] = commissionEmployee;
23          employees[3] = basePlusCommissionEmployee;
24          System.out.printf("Employees processed polymorphically:%n%n");
25          for (Employee currentEmployee : employees) {
26              System.out.println(currentEmployee);
27              // Raise salary for BasePlusCommissionEmployee
28              if (currentEmployee instanceof BasePlusCommissionEmployee) {
29                  // Downcast Employee reference to BasePlusCommissionEmployee reference
30                  BasePlusCommissionEmployee employee =
31                          (BasePlusCommissionEmployee) currentEmployee;
32                  employee.setBaseSalary(1.10 * employee.getBaseSalary());
33                  System.out.printf("New base salary with 10%% increase is: %,.2f%n", employee.getBaseSalary());
34              }
35              System.out.printf("Earned: %,.2f%n%n", currentEmployee.getEarnings());
36          }
37          // Show type name of each object in employees array
38          for (int i = 0; i < employees.length; i++) {
39              System.out.printf("Employee %d is a %s%n", i, employees[i].getClass().getName());
40          }
```

Focused in PayrollSystemTest (ต่อ)

- บรรทัดที่ 25 แสดงคุณสมบัติการมีหลายรูปแบบทำให้อ็อบเจกต์ทุกตัวสามารถอ้างอิงด้วยตัวแปร currentEmployee ซึ่งมีชนิดเป็น Employee ได้
- บรรทัดที่ 28 ใช้ตัวดำเนินการ instanceof เพื่อทดสอบว่าอ็อบเจกต์เป็นอ็อบเจกต์จากคลาสที่ระบุหรือจากซับคลาสของคลาสที่ระบุหรือไม่
- บรรทัดที่ 31 การแปลงจากตัวแปรที่มีชนิดเป็นซูเปอร์คลาสไปเป็นอ็อบเจกต์ที่มีชนิดเป็นซับคลาส (Downcasting)
- บรรทัดที่ 39 การแสดงคลาสของอ็อบเจกต์แต่ละตัวโดยการเรียกเมทอด getClass ซึ่งจะได้ผลลัพธ์เป็นอ็อบเจกต์ชนิด Class ซึ่งเก็บข้อมูลของคลาสของอ็อบเจกต์ที่เราทดสอบ โดยที่อ็อบเจกต์ Class จะมีเมทอด getName ให้เรียกต่อเพื่อให้ได้ชื่อของคลาสมา

Focused in Employee's subclass

- สังเกตุโค้ดส่วนที่ซ้ำกันใน constructor และ method setter ของ subclass

```
4     public SalariedEmployee(String firstName, String lastName, String nationalIdNumber, double salary) {
5         super(firstName, lastName, nationalIdNumber);
6
7         if (salary < 0.0) {
8             throw new IllegalArgumentException("Salary must be >= 0.0");
9         }
10
11        this.salary = salary;
12    }
13
14    public void setSalary(double salary) {
15        if (salary < 0.0) {
16            throw new IllegalArgumentException("Salary must be >= 0.0");
17        }
18
19        this.salary = salary;
20    }
```

Focused in Employee’s subclass (ต่อ)

- ทำไมจึงไม่เรียก setter method จาก constructor ? ทั้ง ๆ ที่โค้ดซ้ำกัน
- Java ไม่อนุญาต ?
  - อนุญาต ให้เรียกได้ แต่!
  - เป็นไปได้ว่าในอนาคตอาจจะมีคลาสอื่นที่สืบทอดจาก SalariedEmployee แล้วไปโอเวอร์ไรด์เมทอด setSalary ซึ่งจะทำให้เกิดปัญหาเมื่อเราต้องการสร้างอ็อบเจกต์จากคลาสใหม่นี้
  - หากต้องการให้ constructor เรียกใช้ method อื่นจริง ควรกำหนดให้ method ที่ถูกเรียกนั้นเป็น static

Final class & Final method

- คลาสที่ถูกประกาศให้เป็น final (Final class) จะไม่สามารถสืบทอดได้
  - เมทอดทุกเมทอดของคลาส final จะถือเป็น final method โดยปริยายเช่นกัน ยกตัวอย่างเช่น คลาส String
- เมทอดที่ถูกประกาศให้เป็น final (Final method) จะไม่สามารถถูกโอเวอร์ไรด์ได้ในซับคลาส
  - เมทอดที่ประกาศเป็น private หรือ static จะถือว่าเป็น final method โดยปริยาย

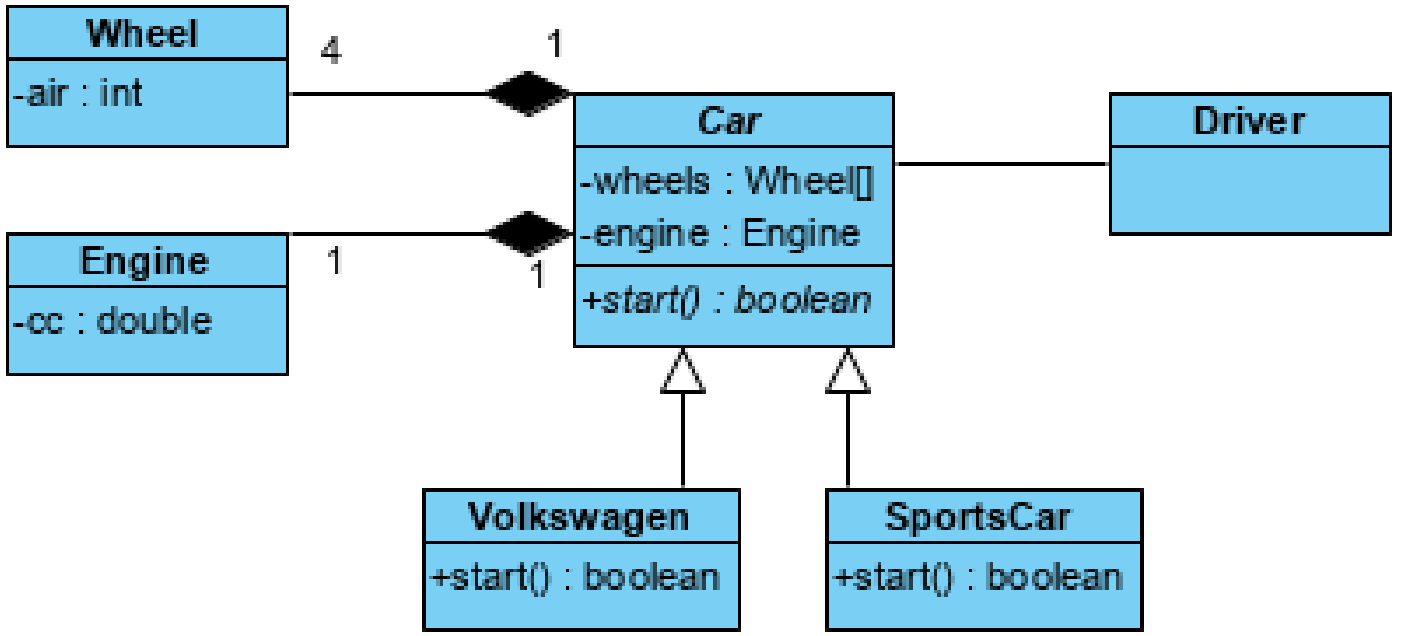
Composition

Has-a relationship

ความสัมพันธ์แบบ has-a

- การที่คลาสหนึ่งมีอ็อบเจกต์ของอีกคลาสหนึ่งเป็นส่วนประกอบหนึ่งของมัน
  - เพื่อการแปลความที่เหมาะสม สามารถใช้คำว่า part-of, member-of แทนคำว่า has-a ได้
- การเป็นส่วนประกอบกันมี 2 ประเภท คือ aggreration และ composition (แต่ในบทเรียนนี้เราจะยังไม่แยกความแตกต่างของทั้งสองประเภท)
- ยกตัวอย่างความสัมพันธ์แบบ has-a คือ Car has-a Wheels และ Car has-a Engine

ความสัมพันธ์แบบ has-a (ต่อ)



Car.java

week06 > Car.java > Car

```java
public abstract class Car {
    private String serie;
    private Wheel[] wheels;
    private Engine engine;

    public Car(String serie, Wheel[] wheels, Engine engine) {
        this.serie = serie;
        this.wheels = wheels;
        this.engine = engine;
    }

    public String getSerie() {
        return this.serie;
    }

    public abstract void start();

    public void showProfile() {
        System.out.println(getSerie() +
        " has-a " + wheels.length +
        " wheels and has-a " +
        engine.getCC() + " of engine");
    }
}
```

Wheel.java

week06 > Wheel.java > Engine

```java
public class Wheel{
    private int air;
    public Wheel(){
        this.air = 30;
    }
}
class Engine{
    private double cc;
    public Engine(double cc){
        this.cc = cc;
    }
    public double getCC(){
        return this.cc;
    }
}
```

Volkswaken.java

week06 > Volkswaken.java > Volkswaken

```java
public class Volkswaken extends Car{
    public Volkswaken(String serie, Wheel[] wheels, Engine engine){
        super(serie, wheels, engine);
    }
    public void start(){
        System.out.println(getSerie()+" is crawling");
    }
}
```

SportsCar.java

week06 > SportsCar.java > SportsCar

```java
public class SportsCar extends Car{
    public SportsCar(String serie, Wheel[] wheels, Engine engine){
        super(serie, wheels, engine);
    }
    public void start(){
        System.out.println(getSerie()+" is started");
    }
}
```

Driver.java

week06 > Driver.java > ...

```java
public class Driver{
    public static void main(String[] args) {
        SportsCar mustang = new SportsCar("Mustang",
                            new Wheel[4], new Engine(4.0));
        mustang.showProfile();
        mustang.start();

        Car beetle = new Volkswaken("Beetle",
                        new Wheel[4], new Engine(2.0));
        beetle.showProfile();
        beetle.start();
    }
}
```

```
E:\playground\fund2\week06>java Driver
Mustang has-a 4 wheels and has-a 4.0 of engine
Mustang is started
Beetle has-a 4 wheels and has-a 2.0 of engine
Beetle is crawling
```

- Polymorphism using interface class

Interface class

- เครื่องมือที่ทำให้สามารถกำหนด เมทอด/ชุดของเมทอด ที่ต้องการให้คลาสต่างชนิดกันได้
- เสมือนเป็นสัญญาว่าคลาสที่ implement ไป จะต้องระบุรายละเอียดให้กับ เมทอด/ชุดของเมทอด ที่ประกาศไว้ใน interface class
- Interface class ไม่มีคุณลักษณะ (ตัวแปร) มีได้เพียงตัวแปรของคลาส (public final static)
- ไม่มีเมทอดที่สมบูรณ์ มีเพียงชื่อเมทอดและลายเซ็นต์ (method signature) เท่านั้น (public abstract...) (ยกเลิกกฎข้อนี้ใน Java SE 8)
- Interface class ไม่สามารถนำมาสร้างอ็อบเจกต์ได้

Interface class (ต่อ)

- แล้วทำไมไม่ใช้ abstract class สำหรับการมีหลายรูป ?
- สาเหตุที่ไม่สามารถใช้ abstract class เพื่อเป็น Superclass ได้ เนื่องจากบางกรณี superclass และ subclass ไม่มีความสัมพันธ์ในเชิง is a เพียงแต่ต้องการความสามารถ/กลุ่มของความสามารถที่เหมือนกันเท่านั้น
  - อีกสาเหตุคือ Java ไม่สามารถทำให้ subclass สืบทอด superclass มากกว่า 1 ตัวได้ (แต่ implement interface class ได้ไม่จำกัด)
- เช่น คลาสใบกำกับสินค้า (Invoice) และ คลาสพนักงาน (Employee) ต่างก็มีความสามารถในการเรียกดู “ยอดค้างชำระ (Payable)” ได้
- ใช้คำว่า interface ในการกำหนดให้คลาสเป็น interface class

ตัวอย่างอินเทอร์เฟซคลาส Payable

```
1    public interface Payable{
2        double getPaymentAmount();
3    }
```

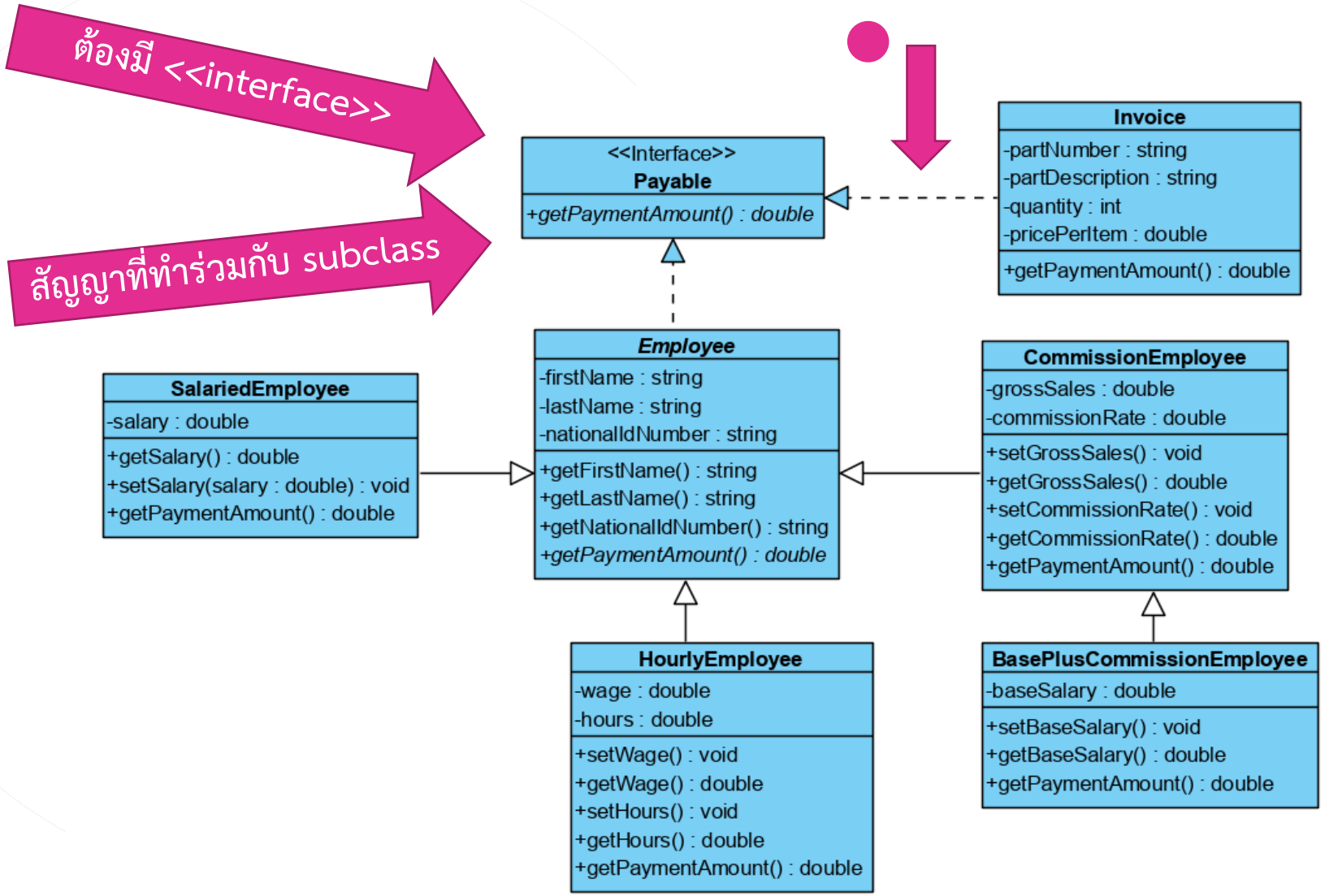
การประกาศคลาสที่อิมพลีเมนต์อินเทอร์เฟซ

```
<class-modifiers> class SubclassName extends SuperclassName
        implements FirstInterface, SecondInterface, ... {
    ...
}
```

```java
public class Invoice implements Payable {
    private final String partNumber;
    private final String partDescription;
    private int quantity;
    private double pricePerItem;

    public Invoice(String partNumber, String partDescription, int quantity, double pricePerItem) {
        if (quantity < 0) {···

        if (pricePerItem < 0.0) {···

        this.partNumber = partNumber;
        this.partDescription = partDescription;
        this.quantity = quantity;
        this.pricePerItem = pricePerItem;
    }

    @Override
    public double getPaymentAmount() {
        return getQuantity() * getPricePerItem();
    }
```

Completed method

Change getEarning() to getPaymentAmount()



Polymorphism using Interface class example

- บรรทัดที่ 12 แสดงการมีหลายรูปผ่าน superclass ที่มีชนิดเป็น interface

```
1   public class PayableInterfaceTest {
2       public static void main(String[] args) {
3           Payable[] payableObjects = new Payable[4];
4
5           payableObjects[0] = new Invoice("01234", "Seat", 2, 10_000.00);
6           payableObjects[1] = new Invoice("56789", "Tire", 4, 2_500.00);
7           payableObjects[2] = new SalariedEmployeeTwo("John", "Smith", "111-11-1111", 100_000.00);
8           payableObjects[3] = new SalariedEmployeeTwo("Lisa", "Barnes", "888-88-8888", 150_000.00);
9
10          System.out.println("Invoices and Employees processed polymorphically:");
11
12          for (Payable currentPayable : payableObjects) {
13              System.out.printf("%n%s%nPayment Due: %,.2f%n", currentPayable, currentPayable.getPaymentAmount());
14          }
15      }
16  }
```

Caution

In rare occasions, a class may implement two interfaces with conflict information

two same constants with different values

two methods with same signature but different return type.

This type of errors will be detected by the compiler.

```
3  public interface Test1
4  {
5      public static final int MAX = 20;
6      public abstract void m1();
7  }
```

```
3  public interface Test2
4  {
5      public static final int MAX = 40;
6      public abstract int m1();
7  }
```

```
3  public class Test implements Test1, Test2
4  {
5      @Override
6      public int m1() {
7          // TODO Auto-generated method stub
8          return 0;
9      }
10 }
```

Error

อินเทอร์เฟซมาตรฐานใน Java

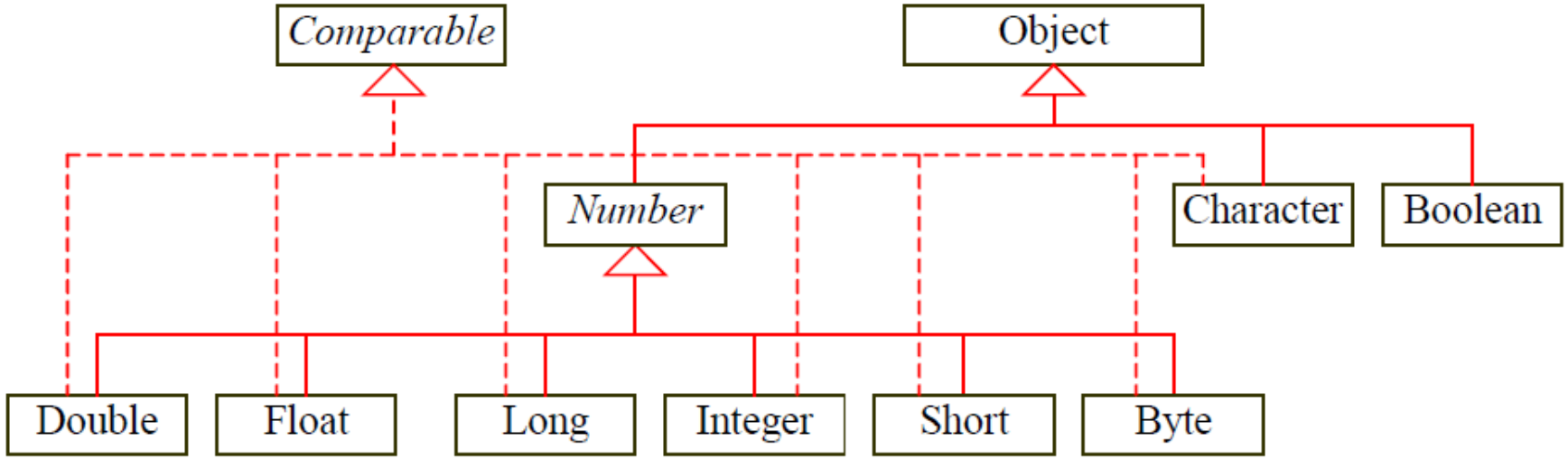
- Comparable คลาสที่ต้องการให้อ็อบเจกต์สามารถเปรียบเทียบลำดับมากกว่าน้อยกว่าได้
- Serializable อิมพลีเมนต์โดยคลาสที่ต้องการให้อ็อบเจกต์สามารถบันทึกลงที่เก็บข้อมูลต่าง ๆ หรือส่งผ่านเครือข่ายได้
- Runnable อินเทอร์เฟซนี้ระบุเมทอด run เอาไว้ และจะอิมพลีเมนต์โดยคลาสที่แทนงานที่ต้องรัน มักจะใช้กับการทำงานแบบหลายเทรดพร้อมกัน (multithreading)
- กลุ่ม GUI event listener การทำงานกับ GUI จะต้องมีการจัดการกับเหตุการณ์ต่าง ๆ ที่เกิดขึ้นจากการที่ผู้ใช้ติดต่อกับส่วนต่อประสานผู้ใช้ เราจะอิมพลีเมนต์อินเทอร์เฟซในกลุ่มนี้เพื่อสร้างตัวจัดการเหตุการณ์

Wrapper Classes

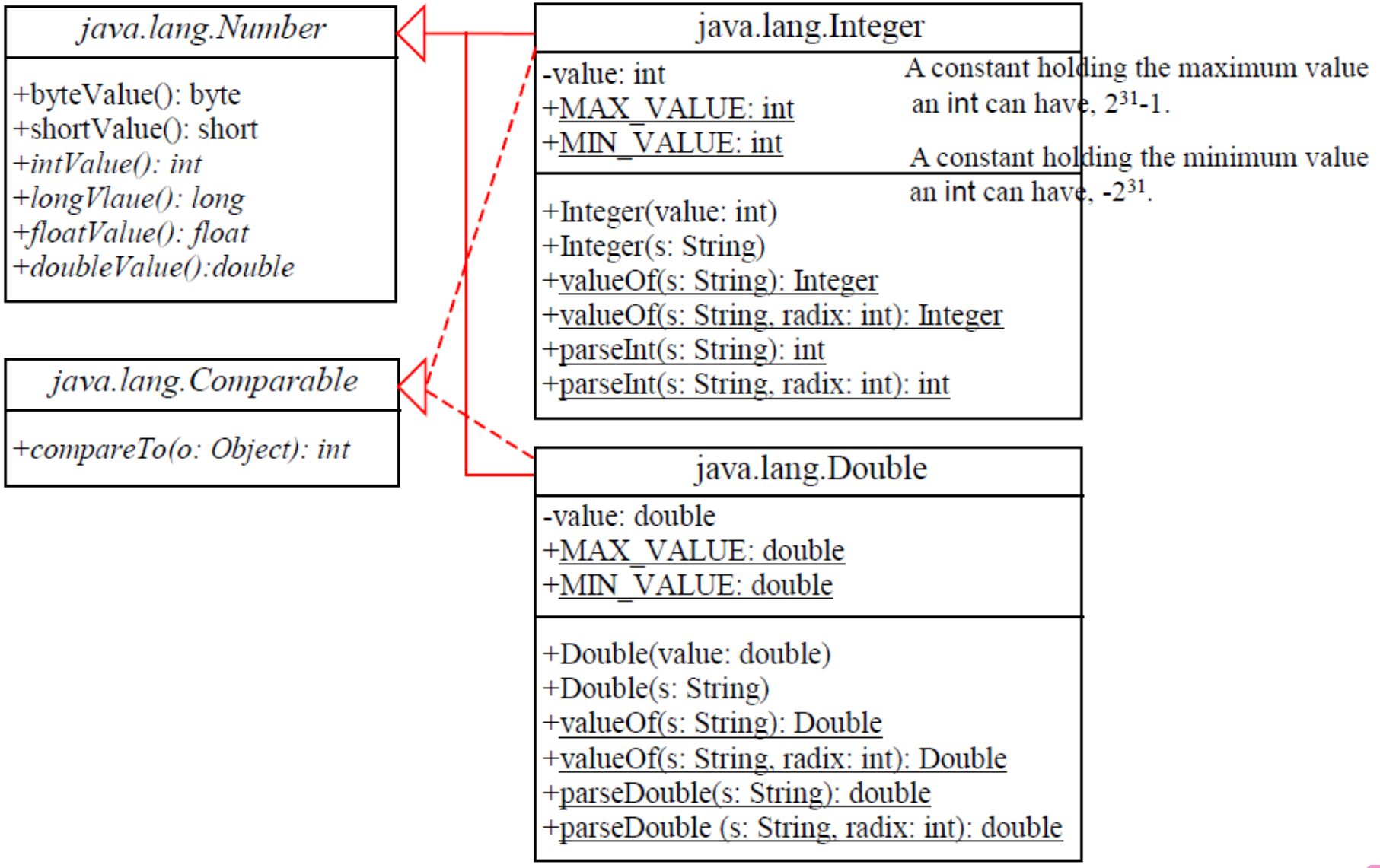
- Boolean
- Character
- Short
- Byte
- Integer
- Long
- Float
- Double

(1) The wrapper classes do not have no-arg constructors.

(2) The instances of all wrapper classes are immutable,



Wrapper Classes



Wrapper Classes

You can construct a wrapper object either from

- a primitive data type value or
- string representing the numeric value.

The constructors for Integer and Double

- public Integer(int value)
- public Integer(String s)
- public Double(double value)
- public Double(String s)

```
Double dObj1 = new Double(8.9);
Double dObj2 = new Double("8.9");
```

Wrapper Classes Cont.

MAX_VALUE / MIN_VALUE

represents the maximum value and minimum value of the corresponding primitive data type, respectively.

```
System.out.println(Integer.MAX_VALUE);
System.out.println(Float.MIN_VALUE);
System.out.println(Double.MAX_VALUE);
```

```
2147483647
1.4E-45
1.79769313486231570E308
```

Conversion Methods

Each numeric wrapper class extends the abstract Number class, which contains

Convert objects into primitive type values.

- doubleValue, floatValue, intValue, longValue
- byteValue and shortValue

```
Double doubleObj = new Double(8.9);
double d = doubleObj.doubleValue();

Integer shortObj = new Integer(8);
short s = shortObj.shortValue();
```

อ้างอิง

- https://nbviewer.jupyter.org/github/Poonna/java-book/