Week7

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

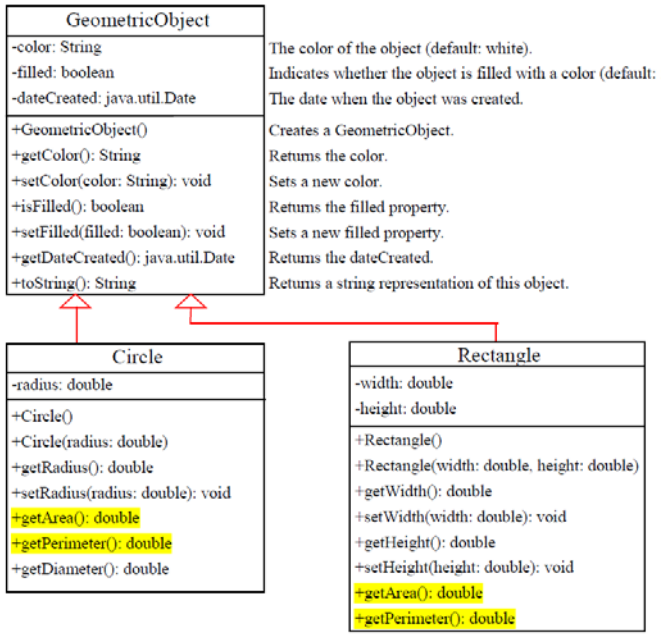
The abstract Modifier

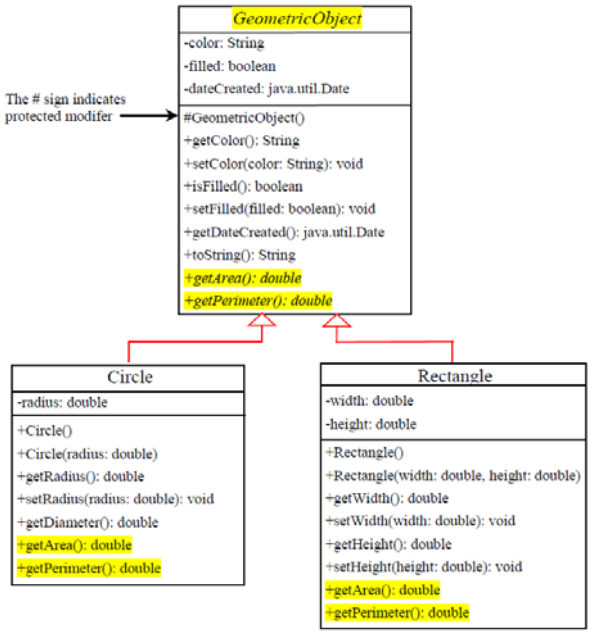
The abstract class

- Cannot be instantiated
- Should be extendedand implemented in subclasses
- `public abstract class Student{... }`

The abstract method

- Method signature without implementation
- `public abstract void m();`





Interface

Programming Fundamentals II

Week 7 **Interface**

1. Interface and Implement
2. Constant in Interface
3. Create Interface with JAVA

Interface

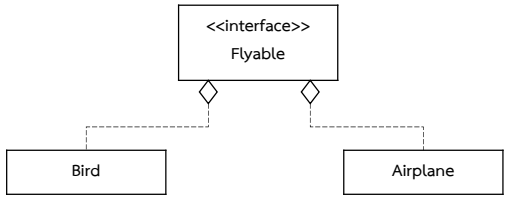
and

Implement

Interface

- Object ที่ไม่ใช่ Class เดียวกัน อาจจะมีความสามารถบางอย่างที่เหมือนกัน
- Ex. Bird & Airplane เป็น Object ที่ต่างประเภทกัน แต่ทั้งสอง สามารถ Fly ได้เหมือนกัน
- จัดให้ Class Bird & Airplane อยู่ใน Group เดียวกัน
- เรียกGroup ของ Class ที่มีความสามารถเหมือนกันว่ามี Interface แบบเดียวกัน

Interface Flyable()




```
interface Flyable
{
    String fly();
}
```

ขาด abstract ??

Implements

```
Class Bird implements Flyable{
    String fly(){
        return “Bird fly”;
    }
}
```

```
Class Airplane implements Flyable{
    String fly(){
        return “Airplane fly”;
    }
}
```


การใช้งาน Interface

1. ประกาศ

```
Flyable f1;
```

2. สร้าง Object

```
f1 = new Bird();
```

3. เรียกใช้ Method ใน Interface

```
System.out.println(f1.fly());
```

การใช้งาน Interface

1. ประกาศ

```
Flyable f1;
```

2. สร้าง Object

```
f1 = new Bird();
```

3. เรียกใช้ Method ใน Interface

```
System.out.println(f1.fly());
```


Interface reference

```
3  public class FlyTest
4  {
5      public static void main(String[] args)
6      {
7          Flyable f1;
8
9          f1 = new Bird();
10
11         f1 = new Airplane();
12
13         System.out.println(f1.fly());
14     }
15 }
```


Interface polymorphism

```
3  public class FlyTest
4  {
5      public static void main(String[] args)
6      {
7          Flyable f1;
8
9          f1 = new Bird();
10
11         System.out.println(f1.fly());
12
13         f1 = new Airplane();
14
15         System.out.println(f1.fly());
16     }
17 }
```

Output:

```
Bird Fly
Airplane Fly
```

- Method เดียวกัน แต่ทำงานแตกต่างกัน

Interface polymorphism

```
3  public interface Flyable
4  {
5      String fly();
6      String Landding();
7  }
```

```
3  public class Bird implements Flyable
4  {
5
6      @Override
7      public String fly() {
8
9          return "Bird Fly";
10     }
11
12 }
```

Interface polymorphism

```
3  public interface Flyable
4  {
5      String fly();
6      String Landding();
7  }
```

```
3  public class Bird implements Flyable
4  {
5      @Override
6      public String fly() {
7          return "Bird Fly";
8      }
9
10     @Override
11     public String Landding() {
12         return "Bird Landding";
13     }
14 }
```


Interface polymorphism

- Bird นอกจากการ Fly แล้ว Bird สามารถ Sing ได้

```
3  public interface Flyable
4  {
5      String fly();
6      String Landding();
7  }
```

```
3  public interface Singable
4  {
5      String Sing();
6  }
```


Interface polymorphism

```
3  public interface Flyable
4  {
5      String fly();
6      String Landding();
7  }
```

```
3  public interface Singable
4  {
5      String Sing();
6  }
```

```
3  public class Bird implements Flyable, Singable
4  {
5      @Override
6      public String Fly() {
7          return "Bird Fly";
8      }
9
10     @Override
11     public String Landding() {
12         return "Bird Landding";
13     }
14
15     @Override
16     public String Sing() {
17         return "Bird Sing";
18     }
19 }
```


String and Date Classes

มีหลาย Classes เช่น String หรือ Date ที่เป็น Class ที่อยู่ใน Java library implement Comparable Interface เพื่อ Override method compareTo

```
public class String extends Object
      implements Comparable {
      public int compareTo(Object o){}
      // class body omitted
}
```

Max Method

ถ้าหาค่าสูงสุดของตัวเลข ใช้ Math class มาช่วยได้

แต่ในการจะทำ Method หาค่าสูงสุดของ Object ทำแบบไหนได้บ้าง

```
// Max.java: Find a maximum object
public class Max {
      public static Comparable max
            (Comparable o1, Comparable o2) {
            if (o1.compareTo(o2) > 0)
                  return o1;
            else
                  return o2;
      }
}
```


Max Method

```
// Max.java: Find a maximum object
public class Max {
       public static Comparable max
              (Comparable o1, Comparable o2) {
              if (o1.compareTo(o2) > 0)
                     return o1;
              else
                     return o2;
       }
}
```

```
String s1 = "f";
String s2 = "e";
String s3 = Max.max(s1, s2);
```

return Comparable

Casting Type

```
String s1 = "f";
String s2 = "e";
String s3 = (String)Max.max(s1, s2);
```


Interfaces vs. Abstract Classes

In an interface, the data must be constants; an abstract class can have all types of data.

Each method in an interface has only a signature without implementation; an abstract class can have concrete methods.

	Variables	Constructors	Methods
Abstract	No restrictions	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator.	No restrictions.
Interface	All variables must be **public static final**	**No constructors**. An interface cannot be instantiated using the new operator.	All methods must be **public abstract** instance methods

Constant

in

Interface

Interface Constant

All data fields are **public final static**

All methods are **public abstract** in an interface.

```
public interface T1 {
  public static final int K = 1;

  public abstract void p();
}
```

Equivalent

```
public interface T1 {
  int K = 1;

  void p();
}
```

ถ้าจะเรียกใช้ ตัวแปร K

Class ที่ Implement Interface เรียกใช้ K ได้เลย

Class ที่ไม่ได้ Implement นี้ต้องเรียกใช้ผ่าน interface

Interface Constant

```
3  public interface Flyable
4  {
5      int absoluteCeiling = 1000;
6
7      String Fly();
8      String Landding();
9  }
```

```
3  public interface Flyable
4  {
5      public static final int absoluteCeiling = 1000;
6
7      String Fly();
8      String Landding();
9  }
```


Interface Constant

Class ที่ Implement Interface Flyable สามารถเรียกใช้ค่าคงที่ได้เลย

ตัวอย่าง **absoluteCeiling**

Class อื่นที่ไม่เกี่ยวข้องกับ Interface(ไม่ได้ Implement) นี้ต้องเรียกเต็มคือ

ตัวอย่าง **Flyable.absoluteCeiling**

Interface Constant

```
3  public class Bird implements Flyable, Singable
4  {
5      @Override
6      public String Fly() {
7          return "Bird Fly below "+ absoluteCeiling;
8      }
9
10     @Override
11     public String Landding() {
12         return "Bird Landding";
13     }
14
15     @Override
16     public String Sing() {
17         return "Bird Sing";
18     }
19 }
```

Interface Constant

```
3  public class FlyTest
4  {
5      public static void main(String[] args)
6      {
7          Flyable f1;
8
9          f1 = new Bird();
10
11         System.out.println(f1.Fly());
12         System.out.println("Ceil : " + Flyable.absoluteCeiling);
13
14         f1 = new Airplane();
15
16         System.out.println(f1.Fly());
17     }
18 }
```


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 }
```

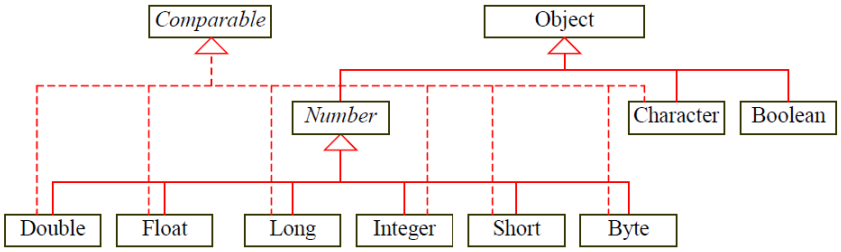
Error

Wrapper Classes

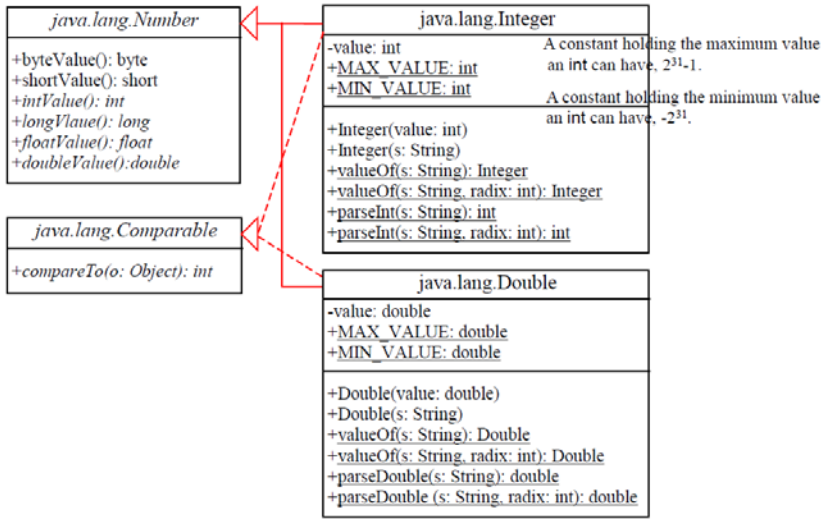
- Boolean
- Character
- Short
- Byte
- Integer
- Long
- Float
- Double

(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
1.7976931348623157E308
```


Conversion Methods

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

Convert objects into primitive type values.

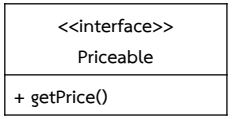
- doubleValue, floatValue, intValue, longValue
- byteValue and shortValue

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

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

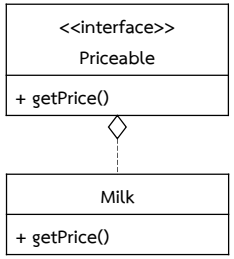

Create Interface with JAVA

EX. Interface Priceable




```
interface Priceable
{
    public int getPrice();
}
```


EX. Class Milk



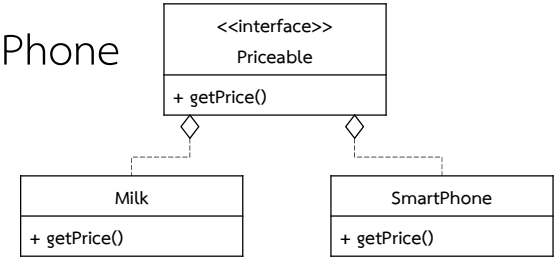

```
class Milk implements Priceable
{
    @Override
    public int getPrice() {
        return 20;
    }
}
```


EX. Class PriceableTest

```
3  public class PriceableTest
4  {
5      public static void main(String[] args)
6      {
7          Priceable item1 = new Milk();
8          System.out.println(item1.getPrice());
9      }
10 }
```

```
20
```


EX. Class SmartPhone



SmartPhone มีการคิดราคาดังนี้

- ถ้าซื้อเครื่องมือ1 มีราคา 25000บาท
- ถ้าใช้ไปแล้ว ราคาจะตกปีละ 2500บาท
- แต่ราคาจะไม่น้อยกว่า 7500บาท

EX. Class SmartPhone

```
3  public class SmartPhone implements Priceable
4  {
5      int originalPrice= 25000;
6      int age = 0;
7
8      @Override
9      public int getPrice()
10     {
11         int deprecation = age * 2500;
12         return Math.max(originalPrice-deprecation, 7500);
13     }
14
15     public void setAge(int year)
16     {
17         age = year;
18     }
19 }
```


EX. Class PriceableTest

```
3  public class PriceableTest
4  {
5      public static void main(String[] args)
6      {
7          Priceable item1 = new Milk();
8          System.out.println(item1.getPrice());
9
10         SmartPhone phone1 = new SmartPhone();
11         phone1.setAge(3);
12
13         Priceable item2 = phone1;
14         System.out.println(item2.getPrice());
15     }
16 }
```

```
20
17500
```


EX. Class Cart

Cart
+ sumPrice()

```
3  public class Cart
4  {
5      Priceable[] item = new Priceable[2];
6
7      public int sumPrice()
8      {
9          int sum = 0;
10         sum = item[0].getPrice()+ item[1].getPrice();
11         return sum;
12     }
13 }
```


EX. Class PriceableTest

```
3  public class PriceableTest
4  {
5      public static void main(String[] args)
6      {
7          Milk item1 = new Milk();
8
9          SmartPhone item2 = new SmartPhone();
10         item2.setAge(3);
11
12         Cart cart = new Cart();
13         cart.item[0] = item1;
14         cart.item[1] = item2;
15
16         System.out.println(cart.sumPrice());
17     }
18 }
```

```
17520
```

Good Luck

Midterm