INFO

Datatype

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Type เริ่มต้นที่ไม่ใช่ class ก็คือ ตัวแปร ไม่ใช่ reference สามารถใช้ = เทียบค่ากันได้

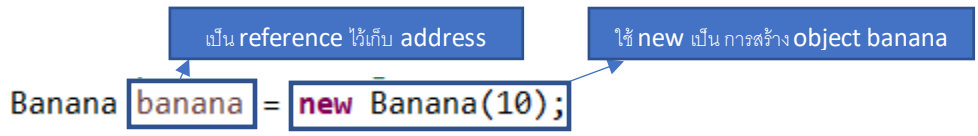
นอกจาก data type ข้างบนนแล้ว ที่เหลือเป็น class ทั้งหมด

CLASS ประกอลด้วย object และ method มั้ง

Class – ต้นแบบ ไม่มีค่าอยู่ข้างใน เช่น ผลไม้

Object - ตัวรถ เช่น ระกระบะ รถเก๋ง

ตัวอย่างการเรียกใช้งาน class แบบ oop จากภาพด้านล่าง



ตัวแปรbananaไม่ใช่ object แต่ เป็น reference เก็บ address เวลาเอาไปใช้ก็ระวังด้วย เช่น String (string เป็นclassนะ)

```
String a = "test";
String b = new String("test");
if(a==b)System.out.println(true);
else System.out.println(false);
```

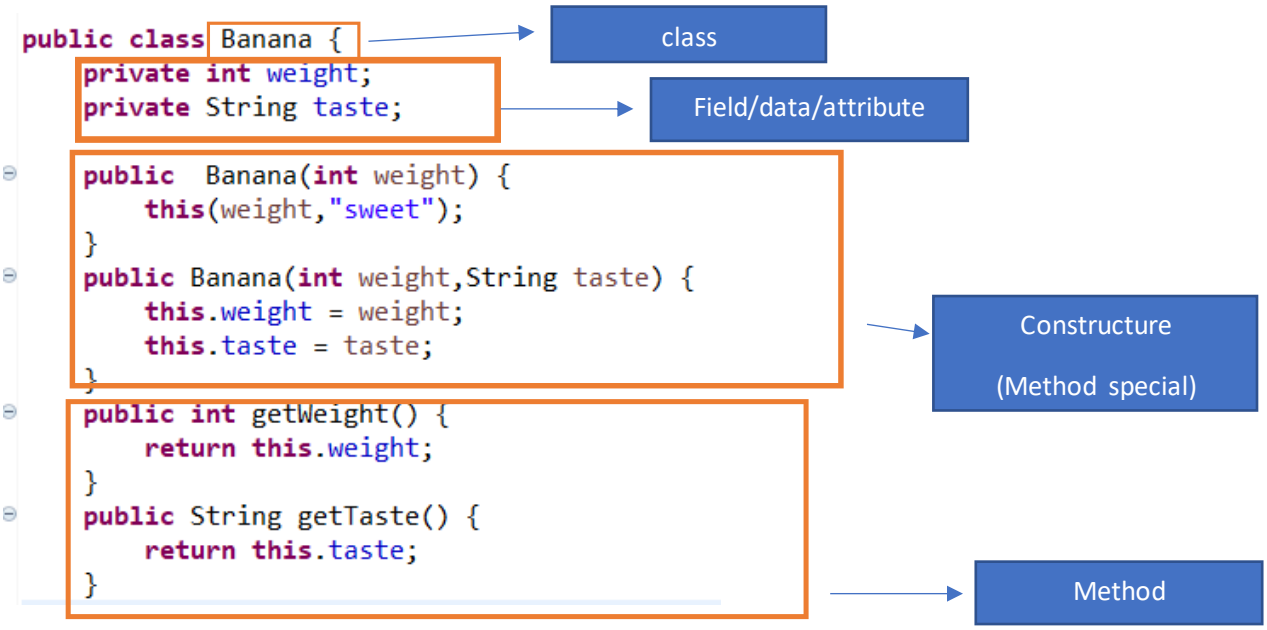
จะได้ false

เพราะ มันเอา address a เทียบ กับ address b ทำให้ผิดพลาด

Class oop มี concept 4 หัวข้อ

1. Encapsulation เช่น

   ประกอบด้วย field และ method (method พิเศษ คือ constructure)



   overloadding

```
public  Banana(int weight) {
    this.weight = weight;
}
public Banana(int weight,String taste) {
    this.weight = weight;
    this.taste = taste;
}
```

Overloading คือ การใช้ชื่อฟังก์ชัน เดิม แต่ parameter ข้างในต่างกัน

```
public  Banana(int weight) {
    this(weight,"sweet");
}
public Banana(int weight,String taste) {
    this.weight = weight;
    this.taste = taste;
}
```

เมื่อไม่อยากเขียนตัวแปรซ้ำกันเยอะๆ ให้ใช้ this เป็นการชี้ address ไปที่ตัวเอง

```
*/
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

อันนี้คือ toString จาก class object คือมันจะมีให้อยู่แล้วในทุก class

```
Banana banana = new Banana(10);
System.out.println(banana);
```

จะ print banana ได้

```
testfruit1.Banana@4617c264
```

ก็คือ ชื่อ package class ตมด้วย @ แล้วปิดด้วย ตัวเลขสักอย่าง (จำไม่ได้ว่าเขาพูดถึงปล่าว)

จากนั้นเขาจะสอน การ override function toString ยกตัวอย่างของ class banana

```
@Override
public String toString() {
    return "weight = " + weight +  "taste = " + taste ;
}
```

กำกับด้วย @Override (จะมีหรือไม่มีก็ได้ เขียนให้รู้ ว่ามีการเขียน function ทับ super class ในที่นี้คือ toString ใน class object) เวลาเรียกใช้ toString หรือ เกิดการ print object นี้ มันจะเรียกใช้ที่เราเขียนไว้แทน จะ print banana ได้

```
weight = 10  taste = sweet
```

```
1  package testfruit1;
2
3  public class Banana {
4      private int weight;
5      private String taste;
6
7      public  Banana(int weight) {
8          this(weight,"sweet");
9      }
10     public Banana(int weight,String taste) {
11         this.weight = weight;
12         this.taste = taste;
13     }
14     public int getWeight() {
15         return this.weight;
16     }
17     public String getTaste() {
18         return this.taste;
19     }
```

อันขวาเพิ่ม tostring

```
package testfruit1;

public class Banana {
    private int weight;
    private String taste;

    public  Banana(int weight) {
        this(weight,"sweet");
    }
    public Banana(int weight,String taste) {
        this.weight = weight;
        this.taste = taste;
    }
    public int getWeight() {
        return this.weight;
    }
    public String getTaste() {
        return this.taste;
    }

    @Override
    public String toString() {
        return "weight = " + weight +  "  taste = " + taste ;
    }
```

2.Inheritance

ก็ คือการสืบทอด มี class แม่ (parent class, super class) และ class ลูก (subclass, child class)

ก็ class ลูก สืบทอด field กับ method มาจาก class แม่ ยกเว้น constructure ไม่สามารถสืบทอดได้ มีของใครของมัน แต่สามารถเรียกใช้ได้ด้วยการใช้ keyword super (เด่วเขียนให้ดูยุข้างล่างหัวข้อ abstract)

สังเกต this เป็นการเรียกใช้เพื่อชี้ไปที่ class ตัวเอง

Super เป็นการเรียกใช้เพื่อชี้ไป class parent

เรื่องการสืบทอดส่วยใหญ่จะเกี่ยวกับ หัวข้อที่ 3 (abstraction)และ 4(polymorphism)

3.abstraction

พูดไม่ถูกอ่ะ มันเป็นที่เอาไว้ใช้แบบของส่วนรวมของ หลายๆclass ที่มีความเกี่ยวข้องกัน ตัวมันไม่สามารถสร้าง object ได้

มี 2 แบบ - คือ abstract class ที่สามารถมี abstract function ซึ่งต้องมีอย่างน้อย 1 อัน

- กับ interface ที่จะถูกเรียกใช้ โดยการ implement ผ่านการ override สามารถimplement หลายอันได้

โดย จะไม่มี field อยู่ข้างใน มีแต่ method และ method จะถูกบังคับใช้ใน class ที่ ถูก implement

แค่มีข้อยกเว้นตรง default method ที่ใน method สามารถมี body และ return ค่าได้

4.polymorphism

ก็ สั้นๆคือ class ลูก สามารถถูก reference ด้วย class แม่ได้

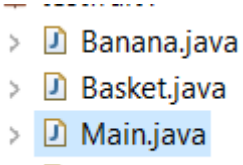
เช่น

```
Banana testba = new Banana(10);
Banana testsub = new subbanana(10,10);
```

Subbanana ถูก extends ด้วย Banana อยู่ แต่ก็สามารถใช้ reference ของ Banana ได้

PRACTICE

Practice 1 File Example 1 หา sum weight ในทุก ผลไม้



Practice จากตอนเช้าที่เขาเฉลยจะมี class อยู่ เราใช้ชื่อไฟล์ example 1 อ่ะ เป็นโจทวันแรกที่เขาให้หา sum

Practice 2 File Example 2

ข้อย่อยละกัน เขาจะให้สร้าง interface ขึ้นมาชื่อ Ediavle (ที่แปลว่าสามารถกินได้อ่ะ) ขึ้นมา แล้วให้ implement Banana ให้ banana สามารถกินได้

```
package testfruit2;

public interface Ediable {
    boolean isPoisioned();
}
```

ใช้โดย

```
package testfruit2;

public class Banana implements Ediable{
    private int weight;
    private String taste;

    public  Banana(int weight) {
        this(weight,"sweet");
    }
    public Banana(int weight,String taste) {
        this.weight = weight;
        this.taste = taste;
    }
    public int getWeight() {
        return this.weight;
    }
    public String getTaste() {
        return this.taste;
    }

    @Override
    public String toString() {
        return "weight = " + weight +  "  taste = " + taste ;
    }
    public boolean isPoisioned() {
        return true;
    }
}
```

Implement Edible ใส่ banana

ต้องเอา method จาก Edible มาใส่ด้วย โดย return true เพื่อเซตว่ามันกินได้

ให้ banana กินได้ แต่ Ispoisioned ใน banana เป็น true ไม่ต้องงงนะ ในห้องเราแค่ตก eng

จากนั้นให้ทำแบบเดิม แต่ ใส่ if print sum weight แค่ที่สามารถกินได้

```
public int sumWeightEdiable() {
    int sum =0;

    for(Banana go : bananaList) {
        if(go.isPoisioned())sum+= go.getWeight();

    }

    return sum;
}
```

ข้อนี้บ่องตงจำไม่ได้ละว่ามีไร และเขาให้ทำไร ข้ามไปก็ได้

แต่ class Ediable สร้างไว้ด้วย

Practice 3 File example 3 ละกัล

ตอนนี้เขาเห็นว่า banana กับ orange มันมีการเขียน weight กับ taste ซ้ำกัน เขาเลยอยากยุบ โดยใช้ abstraction ไว้ที่ fruit

ก็ สร้าง abstract class fruit เมือน orange แหละ แล้ว extends มันไปที่ orange กับ banana

```
abstract public class  Fruit implements Ediable,Matter{
    private int weight;
    private String taste;

    public Fruit(int weight,String taste) {
        this.weight =weight;
        this.taste = taste;
    }
    public int getWeight() {
        return this.weight;
    }
    public String getTaste() {
        return this.taste;
    }
    public boolean isPoisioned() {
        return true;
    }
```

```
public class Banana  extends Fruit{

    public  Banana(int weight) {
        this(weight,"sweet");
    }
    public Banana(int weight,String taste) {
        super(weight,taste);
    }
}
```

```
public class Orange extends Fruit {

    public  Orange(int weight) {
        this(weight,"sour");
    }
    public Orange(int weight,String taste) {
        super(weight,taste);
    }
}
```

แล้วก็ สร้าง class เพิ่มเติม คือ Card เก็บแค่น้ำหนัก กับ สร้าง interface Matter

```
public class Card implements Matter {
    private int weight;

    public Card(int weight) {
        this.weight = weight;
    }
    public int getWeight() {
        return this.weight;
    }
}
```

```
public interface Matter {
    int getWeight();
}
```

จริงๆ fruit กับ Card ตอนแรก ไม่ได้ implement Matter นะ แต่ในรูป มันคือเผลอเขียนไปก่อนละค่อยแคป

โจทแรก 3.1 คือ print sum weight แบบ โจทแรก แต่ใช้ Matter แทน ไปแก้ใน Basket

ลองทำดู ……..

เฉลย

```
public class Basket {
    List<Banana> bananaList = new ArrayList<Banana>();
    List<Orange> orangeList = new ArrayList<Orange>();

    public Basket() {};

    public void add(Banana temp) {
        bananaList.add(temp);
    }
    public void add(Orange temp) {
        orangeList.add(temp);
    }
    // sum weight all
    public int sumWeight() {
        int sum=0;
        for(Banana go : bananaList) {
            sum+= go.getWeight();
        }
        for(Orange go : orangeList) {
            sum+= go.getWeight();
        }
        return sum;
    }
    // sum weight is Edible
    public int sumWeightEdiable() {
        int sum =0;

        for(Banana go : bananaList) {
            if(go.isPoisioned())sum+= go.getWeight();

        }
```

แก้ เป็น

```
public class Basket {
    List<Matter> info = new ArrayList<Matter>();

    public Basket() {};

    public void add(Matter temp) {
        info.add(temp);
    }

    // sum weight all
    public int sumWeight() {
        int sum=0;
        for(Matter go : info) {
            sum+= go.getWeight();
        }
        for(Matter go : info) {
            sum+= go.getWeight();
        }
        return sum;
    }
```

Practice 4 File example 3

จากไฟล์เดิม โจทคือ ให้ print ว่า แต่ละ taste (รสชาติ หนักเท่าไหร่) ก็คือ มีกี่รสก็ print มา แล้วprint น้ำหนักรวมของแต่ละรสมา

ตรงนี้ไปเขียนฟังก์ชั่นเพิ่มใน basket แต่แอบยาก (เพราะเน้น algorithm)

คำใบ้แรก เราจะเข้าถึงตัวแปร ใน fruit โดยการใช้ cast

Cast คล้ายๆการฝืนมั้งเช่น (int)'a' มันจะได้ 97 , (char)98 จะได้ b

ก็เอามาใช้แบบ (fruit)(ชื่อตัวแปร Matter) มันจะออกมาเป็น fruit

คำใบ้ 2 ใช้ instanceof เป็นการเช็คว่า สักอย่าง เช่น

```
if("test" instanceof String)
```

มันจะ return true ถ้า ระบุประเภทถูก

Note ลองใช้ if(5 instanceof int) แล้ว error เลยไปเชิจ คือ instanceof ใช้เช็คได้แต่ class นะ

ลอง ๆ ๆทำดู .....

เฉลย 1

ก็ เริ่มโดยการสร้าง class อันหนึ่ง ให้ข้างใน เก็บ weight และ taste จากนั้น ใน basket ก็สร้าง list ของ classที่พึ่งสร้าง

for loop ไปตาม matter อะแหละ ไล่ดูทุกๆ taste

ถ้า taste ใน matter มีอยู่ใน list class ใหม่ ชื่อ TasteWeight ก็เอา weight มาบวก แต่ถ้าไม่เจอ ก็ add taste กับ weight เข้าไปใหม่ซะ

```java
public class TasteWeight {
    private int weight;
    private String taste;
    public TasteWeight(int weight,String taste) {
        this.weight = weight;
        this.taste = taste;
    }
    public int getWeight() {
        return this.weight;
    }
    public String getTaste() {
        return this.taste;
    }
    public void plusWeight(int plus) {
        this.weight+= plus;
    }
    @Override
    public String toString() {
        return "weight = " + weight + " taste = " + taste ;
    }
}
```

```java
public List<TasteWeight> findTasteWeight(){
    List<TasteWeight> anser = new ArrayList<TasteWeight>();

    for(Matter temp : info) {
        if(temp instanceof Ediable && temp instanceof Fruit) {// (check object temp have
            int tempWeight = ((Fruit)temp).getWeight(); // use cast change temp is fruit
            String tempTaste = ((Fruit)temp).getTaste();  // use cast change temp is frui

            int i =findPosition(anser,tempTaste); // find taste in temp    have in taste
            if(i<0) { // if dont have
                anser.add(new TasteWeight(tempWeight,tempTaste)); // add new taste and ne
            }
            else { // if have
                anser.get(i).plusWeight(tempWeight); // plus another weight in same taste
            }
        }
    }

    return anser;
}
public int findPosition(List<TasteWeight> anser,String taste) {// check taste have taste
    for(int i=0 ; i<anser.size() ; i++) {
        if(taste.equals(anser.get(i).getTaste()))return i;
    }
    return -1;
}
```

```java
for(TasteWeight run : test1.findTasteWeight()) {
    System.out.println(run);
}
```

```
weight = 11 taste = sweet
weight = 9 taste = sour
```

แล้วก็ return array list มา print

เฉลย 2

อันนี้พี่เขากั๊กไว้ เพ่งบอกตอนเลิก ว่า ใช้ map แทน arraylist จะง่ายกว่าเยอะ (เพราะขั้นตอนหา taste ใน TasteWeight ต้อง for loop ไล่ไป แต่ map สามารถระบุได้เลย อย่างโกง)

```java
public HashMap findTasteWeightByMap() {
    HashMap<String,Integer> anser = new HashMap();
    for(Matter temp : info) {
        if(temp instanceof Ediable && temp instanceof Fruit) {// (check object temp have Ed
            int tempWeight = ((Fruit)temp).getWeight(); // use cast change temp is fruit fi
            String tempTaste = ((Fruit)temp).getTaste();  // use cast change temp is fruit

            if(anser.containsKey(tempTaste)) { // search temptase in anser
                int debug = tempWeight+anser.get(tempTaste);
                anser.replace(tempTaste,debug);
            }
            else {
                anser.put(tempTaste, tempWeight);
            }
        }
    }
    return anser;
```

ได้

```java
}
System.out.println(test1.findTasteWeightByMap());
```

```
{sweet=11, sour=9}
```

ลืมมมม อันนี้ดีกว่า

```java
public void add(Banana temp) {
    bananaList.add(temp);
}
public void add(Orange temp) {
    orangeList.add(temp);
}
```

Basket

```java
//add banana
Basket test1 = new Basket();
test1.add(new Banana(1));
test1.add(new Banana(2));
test1.add(new Banana(3));
test1.add(new Banana(2));
test1.add(new Banana(1));
test1.add(new Banana(2));

//add orange
test1.add(new Orange(1));
test1.add(new Orange(2));
test1.add(new Orange(3));
test1.add(new Orange(2));
test1.add(new Orange(1));
```

Main

จริงๆเขียน add ทับกันได้ ด้วยกฎ overloading สวยงามกว่าเยอะ แต่ลืม ขี้เกียจกลับไปแก้ละ ใน flie example1, example2