LESSON VI.
Aggregation and Inheritance

Vu Thi Huong Giang
SoICT (School of Information and Communication Technology)
HUST (Hanoi University of Science and Technology)

Objectives

- Understand the aggregation and inheritance relationships among classes and objects

Content

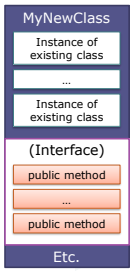
- Aggregation
  - Principles
  - Order of initialization
- Inheritance
  - Principles
  - Inheritance hierarchy
  - Sub class definition
    - extends
  - Order of initialization
    - super

I. AGGREGATION

1. Principles
2. Order of initialization

1. Principle

- Reusing through object:
  - Create new functionality: taking existing classes and combining them into a new whole class
  - Create an interface comprising of public methods for this new class for interoperability with other code
- Relation
  - Existing class is a part of new whole class
  - New whole class has an existing class
  - → Reuse attributes and operations of existing classes through the instances of these classes.

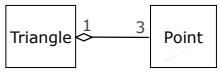


Example: Point – an existing class

```
public class Point {
    private int x; // x-coordinate
    private int y; // y-coordinate
    public Point(){}
    public Point(int x, int y) {
      this.x = x;
      this.y = y;
    }
    public void setX(int x){ this.x = x; }
    public int getX(){ return x; }
    public void setY(int y){ this.y = y; }
    public int getY(){ return y; }
    public void displayPoint(){
      System.out.print("(" + x + "," + y + ")");
    }
}
```

Triangle – whole class

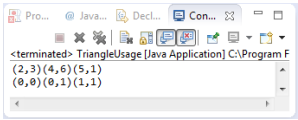
```
public class Triangle {
    private Point d1, d2, d3;
    public Triangle(Point p1, Point p2, Point p3){
      d1 = p1; d2 = p2; d3 = p3;
    }
    public Triangle(){
        d1 = new Point();
        d2 = new Point(0,1);
        d3 = new Point (1,1);
    }
    public void displayTriangle(){
      d1.displayPoint();
      d2.displayPoint();
      d3.displayPoint();
      System.out.println();
    }
}
```



7

Triangle - usage

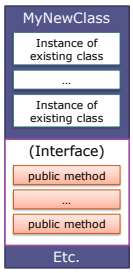
```
public class TriangleUsage {
    public static void main(String[] args) {
        Point d1 = new Point(2,3);
        Point d2 = new Point(4,6);
        Point d3 = new Point (5,1);
        Triangle triangle1 = new Triangle(d1, d2, d3);
        Triangle triangle2 = new Triangle();
        triangle1.displayTriangle();
        triangle2.displayTriangle();
    }
}
```



8

2. Order of initialization

- Initialize all instances of reused existing classes
  - Call their constructors
- Initialize an instance of the whole class
  - Call its constructor



INHERITANCE

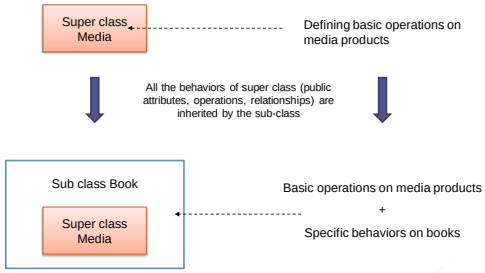
1. Principles
2. Subclass definition

1. Principles

- Reusing through class: create a new class by extending the functionality of an existing class
  - The existing class is called the parent class, or super class, or base class
  - The new class is called the child class, or subclass, or derived class
- Relation: New class is a kind of existing class
  - A subclass inherits the operations, attributes and hierarchical relationships of its super class
    - If a method is defined in a super class, all of its sub class inherit automatically this method
    - If a set of member variables are defined in a super class, all of its sub classes inherit the same set of member variables
- To provide new functionality to a subclass, we can
  - Define new methods and variables for this subclass only
  - Override (execute instead of) methods of the super class in this subclass

11

Example



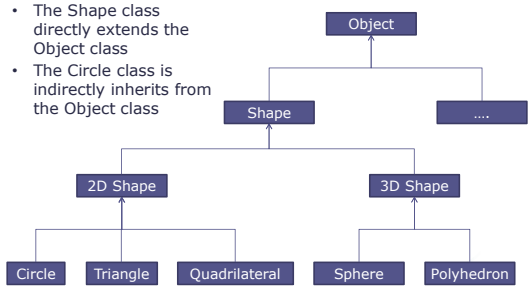
12

2. Class hierarchy

- The class hierarchy defines the inheritance relationship between classes.
  - The root of the class hierarchy is the class Object.
  - Every class in Java directly or indirectly extends (inherits from) this class.
- Direct super class: the super class from which a subclass explicitly inherits.
  - Java: a class can only have one direct super class (single inheritance)
- Indirect super class: any class above the direct super class in the class hierarchy
- The constructors and destructors are not inherited.

Example

- The Shape class directly extends the Object class
- The Circle class is indirectly inherits from the Object class



Visibility of inherited members

- A public member is accessed from any class
- A super classes protected members can be accessed by members of its subclasses and by members of classes in the same package.
- A private member is only accessible from inside the class in which it was declared.

	Public member	Protected member	Private member	Default member
Inside class	Yes	Yes	Yes	Yes
subclasses inside package	Yes	Yes	No	Yes
subclasses outside package	Yes	Yes	No	No
classes with non-inheritance relationship outside package	Yes	No	No	No

3. Subclass definition

```
[access-modifier] class Subclass-name
      extends Superclass-name{
         //       New added variables
         //       New methods
         //       Overridden methods
}
```

16

Example

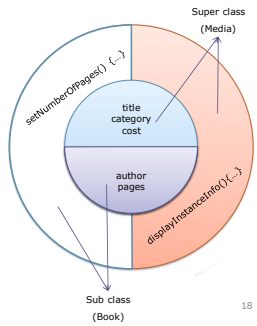
```
// Media class (super class)
public class Media {
    protected String title;
    protected String category;
    protected float cost;
...
    public void  displayInstanceInfo() {
    ...
    }
...
}
```

```
// Book class (sub-class)
public class Book extends Media {
    protected String author[];
    protected int pages;
    public Book() {
    }
// new methods
    public void setNumberofPages(int pages){
    this.pages = pages;
    }
// overridden method
    public void displayInstanceInfo() {
    //....
    }
}

// Usage
public class BookClassUsage {
public static void main(String[] args) {
    Book obj = new Book();
    obj.displayInstanceInfo();
}
}
```

17

Instance of subclass

- An instance of a subclass comprises of:
  - member variables and methods that were defined by the super class
  - member variables and methods that were added by the subclass



18

Example: Account – super class

```
class Account {
    // Member variables
    protected String owner; // Account name
    protected long balance; // Balance
    // value setting Methods
    public void setData(String name, long init_balance) {
       owner = name;
       balance = init_balance;
    }
    public void display() {
       System.out.print("Owner:" + owner);
       System.out.println("\t Balance:" + balance);
    }
}
```

Example: ChargeAccount - subclass

```
public class ChargeAccount extends
    Account{
    // Additional member variables
    private int overdraft;
    // borrowing amount
    private int overdraft_limit;
    /* limit to the amount withdrawn
    * from an account even though
    * there is zero balance */

    // Additional methods
     public void setOverdraftLimit(int
        overdraft_limit) {
          this.overdraft_limit =
            overdraft_limit;
          this.overdraft = 0;
    }
    public void loan(int overdraft) {
         int current_overdraft =
           this.overdraft + overdraft;
         if (current_overdraft <=
           overdraft_limit)
         this.overdraft += overdraft;
         else System.out.println("The
           limit to the amount
           withdrawn is exceeded !!!
           ");
    }
    // overridden method
    public void display() {
         System.out.println("\t\t
           Borrowing amount limit:"+
           overdraft_limit);
         System.out.println("\t\t
           Borrowing amount:" +
           overdraft);
    }
}
```

Example: ChargeAccountClassUsage

```
public class ChargeAccountClassUsage {
    public static void main(String[] args) {
        // creat a super class object
        Account account = new Account();
        // create a sub class object
        ChargeAccount chargeacc = new ChargeAccount();
        // sub class object calls methods from its super class: ok
        chargeacc.setData("Giang", 1000000);
        chargeacc.setOverdraftLimit(1000);
        chargeacc.loan(2000);
        // super class object calls methods from its own class: ok
        account.setData("Tuan", 2000000);
        // super class object calls methods from its sub class: no
        account.setOverdraftLimit(1000);
        // can not call method from its super class, once it is overridden
        chargeacc.display();
        ((Account) chargeacc).display();
    }
}
```

4. Initialization order

- Objects are constructed top-down under inheritance.
- Super class is initialized before its subclass
  - first call super-class constructors
  - then call sub-class constructors
- The constructors of the direct super class are always called
  - Explicit call: call the super class constructor from the sub class constructor code
  - Implicit call: call the parameter-less or default constructor of the super class (if present); no call is written in the code
- Each super class' constructor manipulates the instance variables that the subclass will inherit

22

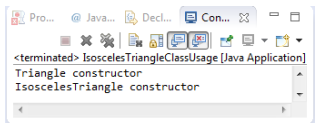
Implicitly call of super class constructor: Example 1

```
public class Triangle {
    private Point d1, d2, d3;
    public Triangle
        (Point p1, Point p2, Point p3){
        d1 = p1; d2 = p2; d3 = p3;
    }
    public Triangle(){
        System.out.println("Triangle constructor");
        d1 = new Point();
        d2 = new Point(0,1);
        d3 = new Point (1,1);
    }
    public void displayTriangle(){
        d1.displayPoint();
        d2.displayPoint();
        d3.displayPoint();
        System.out.println();
    }
}
```

23

Implicitly call of super class constructor: Example 1

```
public class IsoscelesTriangles extends Triangle{
    public IsoscelesTriangles() {
        // automatic call Triangle()
        System.out.println("IsoscelesTriangle constructor");
    }
}

public class IsoscelesTriangleClassUsage {
    public static void main(String[] args) {
        IsoscelesTriangles obj = new IsoscelesTriangles();
    }
}
```



Implicitly call of super class constructor: Example 2

```
public class Triangle {
    private Point d1, d2, d3;
    public void displayTriangle(){
        d1.displayPoint();
        d2.displayPoint();
        d3.displayPoint();
        System.out.println();
    }
}
```

```
public class IsoscelesTriangles extends
Triangle{
    public IsoscelesTriangles() {
        // automatic call Triangle();
        System.out.println
        ("IsoscelesTriangle constructor");
    }
}

public class IsoscelesTriangleClassUsage {
    public static void main(String[] args) {
        IsoscelesTriangles obj =
        new IsoscelesTriangles();
    }
}
```


Implicitly call of super class constructor: Example 3

```
public class Triangle {
    private Point d1, d2, d3;
    public Triangle
        (Point p1, Point p2, Point p3){
        System.out.println
            ("Triangle constructor");
        d1 = p1; d2 = p2; d3 = p3;
    }
    public void displayTriangle(){
        d1.displayPoint();
        d2.displayPoint();
        d3.displayPoint();
        System.out.println();
    }
}
```

```
public class IsoscelesTriangles extends
Triangle{
    public IsoscelesTriangles() {
        // automatic call Triangle();
        System.out.println
        ("IsoscelesTriangle constructor");
    }
}

public class IsoscelesTriangleClassUsage {
    public static void main(String[] args) {
        IsoscelesTriangles obj =
        new IsoscelesTriangles();
    }
}
```

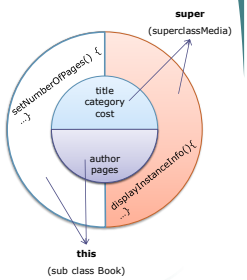
Failed, because class Triangle has user-defined constructor, but no constructor Triangle() is found in the super class.

Explicit call of super class constructor

- If the super class defines explicitly his constructors with parameter, the sub class must call them explicitly
  - Sub class constructor (either with or without parameters) calls the super class constructor in his first statement using **super** keyword.
- The sub class could also call explicitly the constructor without parameter of its super class
- Depending on the passed argument when a constructor make a super call, the relevant constructor of super class is executed.

5. super keyword

- This keyword is used to indicate the super class of the caller class
- Two usages of **super**:
  - to call the super-class constructor
    super(parameter list);
  - to access super-class members
    super.variable;
    super.method(parameter list);

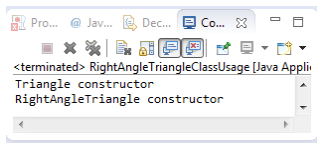


Call super class' constructor from a constructor without parameter

```
public class Triangle {
    private Point d1, d2, d3;
    public Triangle(){
        d1 = new Point(); d2 = new Point(0,1);
        d3 = new Point (1,1);
    }
    public Triangle(Point p1, Point p2, Point p3){
        System.out.println("Triangle constructor");
        d1 = p1; d2 = p2; d3 = p3;
    }
    public void displayTriangle(){
        d1.displayPoint();
        d2.displayPoint();
        d3.displayPoint();
        System.out.println();
    }
}
```

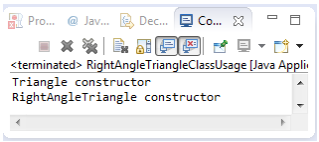
Call super class' constructor from a constructor without parameter

```
public class RightAngleTriangle extends Triangle{
    public RightAngleTriangle() {
        // explicitly call the constructor with parameter
        // of super class
        super(new Point(0,0), new Point(1,0), new Point(0,2));
        System.out.println("RightAngleTriangle constructor");
    }
}
public class RightAngleTriangleClassUsage {
    public static void main(String[] args) {
        RightAngleTriangle obj = new RightAngleTriangle();
    }
}
```



Call super class' constructor from a constructor with parameter

```
public class RightAngleTriangle extends Triangle{
    public RightAngleTriangle(Point p1, Point p2, Point p3) {
        // explicitly call the constructor with parameter of super class
        super(p1, p2, p3);
        System.out.println("RightAngleTriangle constructor");
    }
}
public class RightAngleTriangleClassUsage {
     public static void main(String[] args) {
        RightAngleTriangle obj1 = new
            RightAngleTriangle(new Point(0,0), new Point(1,0), new Point(0,2));
     }
}
```



Access super class' members

- The super keyword allows accessing to the hidden variables or methods of the super-class
- When a sub-class declares the variables or methods with the same names and types as its super-class, the re-declared variables or methods hide those of the super-class.

Access super class' method

```
// Account class
class Account {
     // Member variables
     protected String owner;
     protected long balance;

     //…

     public void display() {
     System.out.print("Owner:" + owner);
     System.out.println("\t Balance:" +
     balance);
     }
}
```

```
public class ChargeAccount extends Account{
     // Additional member variables
     private int overdraft;
     private int overdraft_limit;
     //…
     // access to the super class' member

     public void displayInfo() {
     // access super class method;
        super.displayInfo();

        System.out.println("\t\t Borrowing
        amount limit:"+ overdraft_limit);
        System.out.println("\t\t Borrowing
        amount:"+ overdraft);
     }
}
```

Access super class' variable

```
// Account class
class Account {
     // Member variables
     protected String owner;
     protected long balance;

     //…

     public void display() {
     System.out.print("Owner:" + owner);
     System.out.println("\t Balance:" +
     balance);
     }
}
```

```
public class ChargeAccount extends Account{
     // Additional member variables
     private int overdraft;
     private int overdraft_limit;
     //…
     // access to the super class' member

     public void displayInfo() {
     // access super class method
     // super.displayInfo();

     // access directly to super class's variables
     // to do the same thing
//System.out.print("Owner:" + super.owner);
//System.out.println("\t Balance:" + super.balance);

        System.out.println("\t\t Borrowing
        amount limit:"+ overdraft_limit);
        System.out.println("\t\t Borrowing
        amount:"+ overdraft);
     }
}
```

6. Final modifier

- The final modifier indicates a class that can not be sub-classed.
  - Example

```
public final class EquilateralTriangle extends
  Triangle{
  //
  }
```

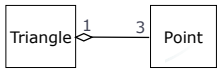
- If a method is declared with the final modifier, it can not be overridden
- If a variable is declared with the final modifier, it can not be changed after its initialization

Quiz 1: Aggregation

- Implement the class Triangle in a different way
  - Hint : using the array data type.

Quiz 1: Aggregation
Another implementation of Triangle

```
public class AnotherTriangle {
    private Point[] Point = new Point[3];
    public AnotherTriangle(Point p1, Point p2, Point p3){
        Point[0] = p1;
        Point[1] = p2;
        Point[2] = p3;
    }
    public void displayTriangle(){
        Point[0].displayPoint();
        Point[1].displayPoint();
        Point[2].displayPoint();
        System.out.println();
    }
}
```



37

Quiz 2: Inheritance

- Modify the Account and ChargeAccount class to visualize following cases
  - Encapsulation of the Account class
  - Call the constructor of its own class
  - Call the constructor of its super class
  - Hint: using private, this and super keywords

Quiz 2: encapsulation

```
//Account class
public class Account {
    // Member variables
    private String owner; // Account name
    private long balance; // Balance

    // value setting methods
    public void setOwner(String name) {
        this.owner = name;
    }
    public void setBalance(long balance){
        this.balance = balance;
    }

    // value getting methods
    public String getOwner(){
        return this.owner;
    }
    public long getBalance(){
        return this.balance;
    }
}
```

Quiz 2:
call the constructor of its own class

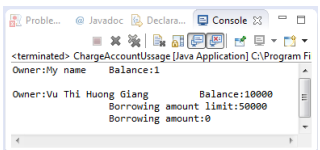
```
//Account class
public class Account {
    // Member variables
    private String owner; // Account name
    private long balance; // Balance

    // constructor with parameters
    public Account (String owner, long balance){
        this.owner = owner;
        this.balance = balance;
    }

    public Account() {
        // constructor call of own class
        this("My name", 1);
    }

    public void display() {
        System.out.print("Owner:" + owner);
        System.out.println("\t Balance:" + this.balance);
    }
}
```

Quiz 2: call the constructor of its super class

```
public class ChargeAccount extends Account{
    // Additional member variables
    private int overdraft; // borrowing amount
    private int overdraft_limit;

    //constructor of sub-class
    public ChargeAccount(String name, long balance, int overdraft_limit){
        // explicit call to super class;
        super(name, balance);
        this.overdraft = 0;
        this.overdraft_limit = overdraft_limit;
    }

    public void display() {
        // access to the super class' method
        super.display();
        System.out.println("\t\t Borrowing amount limit:"+ overdraft_limit);
        System.out.println("\t\t Borrowing amount:" + overdraft);
    }
}
```

Quiz 2: this and super

```
public class ChargeAccountUssage {
    public static void main(String[] args) {
        // Use the constructor without argument
        Account obj = new Account();
        obj.display();
        System.out.println();

        // For sub-class method, use superclass method
        ChargeAccount obj2 = new ChargeAccount("Vu Thi Huong Giang",10000,50000);
        obj2.display();
    }
}
```



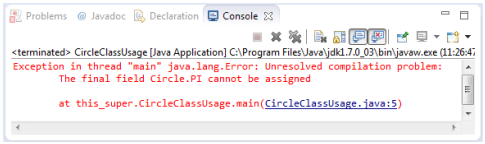
Quiz 3: Final modifier

```
public class Circle {
    public static final double PI = 3.14159;
    public double x, y;
    public double r;
}


public class CircleClassUsage {
public static void main(String[] args) {
    Circle.PI = 4;
}
```

Is this correct ?

Quiz 3: final

- The final keyword means that this variable can never be changed.



Review

- Aggregation:
  - Create new functionality by taking other classes and combining them into a new class.
  - Formulate an common interface to this new class
- Inheritance:
  - Extend the functionality of a class by creating a subclass.
  - Override super class members in the subclasses to provide new functionality