TRƯỜNG ĐẠI HỌC

GIAO THÔNG VẬN TẢI

University of Transport and Communications

HỌC PHẦN: CÔNG NGHỆ JAVA

Người thực hiện: Đào Thị Lệ Thủy

Chương 3. OOP – LỚP VÀ ĐỐI TƯỢNG TRONG JAVA

KHÁI NIỆM VỀ KẾ THỪA VÀ ĐA HÌNH

1. Kế thừa trong Java

- Kế thừa (Inheritance) là một trong những đặc tính quan trọng nhất trong OOP, nó cho phép xây dựng mối liên quan giữa hai lớp với nhau, trong đó có lớp cha (*Superclass*) và lớp con (*Subclass*).

  - Ví dụ về lớp cha và các lớp con

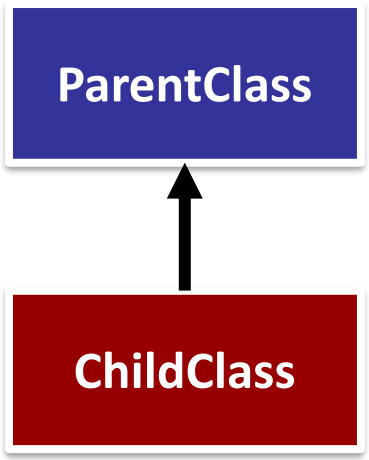
Superclass	Subclass
Shape	Rectangle, Oval, Diamond
Person	Student, Teacher, Worker, Manager
Animal	Dog, Cat, Duck

1. Kế thừa trong Java

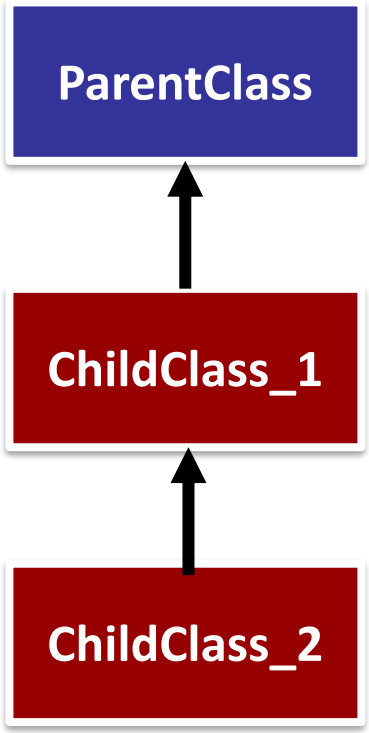
- Khi kế thừa, lớp con sẽ được:
  - ***Thừa hưởng toàn bộ các thuộc tính và các phương thức*** *mà lớp cha cho phép*.
  - ***Định nghĩa thêm các thuộc tính và các phương thức*** của riêng nó.
  - ***Định nghĩa lại các phương thức*** để phù hợp với lớp con.
- **Lợi ích của kế thừa:**
  - Tái sử dụng lại code – ***reusability***.
  - Cho phép tính mở của chương trình – các thay đổi chỉ tác động đến lớp con, không ảnh hưởng đến lớp cha cũng như các phần cốt lõi của chương trình – ***extensibility***.
  - Dễ dàng tiếp cận cho việc thiết kế ban đầu và mở rộng thiết kế sau này – ***designability***.

1. Kế thừa trong Java

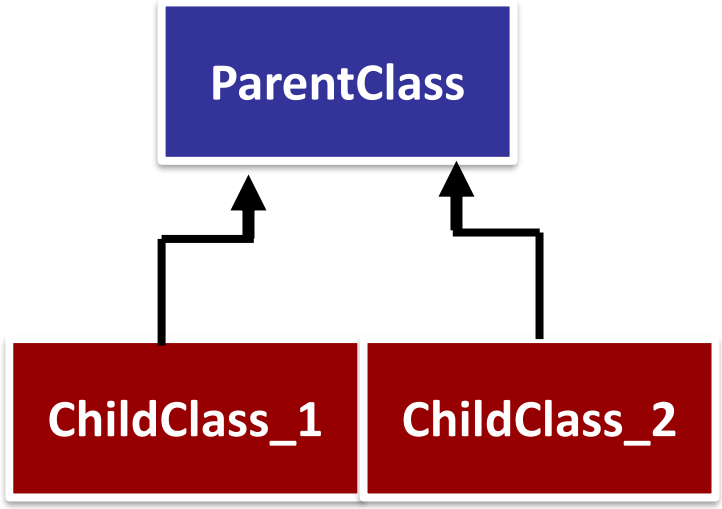
- Trong Java, chỉ hỗ trợ đơn kế thừa mà không hỗ trợ đa kế thừa. Đa kế thừa trong Java được thiết kế dựa trên khái niệm ***interface***.



1) Single



2) MultiLevel



3) Hierarchical

1. Kế thừa trong Java

Cú pháp tạo lớp kế thừa trong Java:

```
public ParentClass {
…
}
public ChildClass extends ParentClass{
…
}
```

Chú ý: *việc truy xuất vào các thuộc tính và phương thức của lớp cha phụ thuộc vào phạm vi truy cập mà lớp cha định nghĩa.*

	ACCESS LEVELS			
MODIFIER	Class	Package	Subclass	Everywhere
public	Y	Y	Y	Y
protected	Y	Y	Y	N
default	Y	Y	N	N
private	Y	N	N	N

1. Kế thừa trong Java

```
public ParentClass {
…
}
public ChildClass_1 extends ParentClass{
…
}
public ChildClass_2 extends ChildClass_1{
…
}
```

2) MultiLevel

```
public ParentClass {
…
}
public ChildClass_1 extends ParentClass{
…
}
public ChildClass_2 extends ParentClass{
…
}
```

3) Hierarchical

1. Kế thừa trong Java – Ví dụ

- **Kế thừa Single**

```
public class Parent{
    public void color(){
        System.out.println("màu xanh");}
}
```

```
public class Child extends Parent {
    public void show(){
        System.out.print("Đây là quả táo ");}
}
```

```
public class Test {
    public static void main(String args[]) {
        Child c = new Child();
        c.show();
        c.color();
    }
}
```

Kết quả

```
Đây là quả táo màu xanh
```

1. Kế thừa trong Java

❑ Ví dụ kế thừa MultiLevel

```
public class Parent{
    public void color(){
      System.out.println("màu xanh");}
}
```

```
public class Child1 extends Parent {
    public void show1(){
       System.out.print("Đây là quả táo ");}
}
```

```
public class Child2 extends Child1 {
    public void show2(){
      System.out.println("Xuất xứ Việt
    Nam");}
}
```

```
public class Test {
    public static void main(String
     args[]) {
Child2 c2 = new Child2();
        c2.show1();
        c2.color();
        c2.show2();
    }
}
```

Kết quả

```
Đây là quả táo màu xanh
Xuất xứ Việt Nam
```

1. Kế thừa trong Java

❑ Ví dụ kế thừa Hierarchical

```
public class Parent{
    public void color(){
      System.out.println("màu xanh");}
}
```

```
public class Child1 extends Parent {
    public void show1(){
      System.out.print("Đây là quả táo ");}
}
```

```
public class Child2 extends Parent {
    public void show2(){
      System.out.print("Đây là quả cam");}
}
```

```
public class Test {
    public static void main(String
     args[]) {
      Child1 c1 = new Child1();
       c1.show1();
       c1.color();

      Child2 c2 = new Child2();
       c2.show2();
       c2.color();
   }
}
```

Kết quả

```
Đây là quả táo màu xanh
Đây là quả cam màu xanh
```

1. Kế thừa trong Java

Một số từ khóa được sử dụng:

- **this**: dùng để tham chiếu đến instance của lớp hiện tại để thông qua đó có thể truy cập thuộc tính và phương thức của đối tượng.
- **this():** để gọi đến constructor từ một constructor khác của lớp
- **this**: cũng có thể là một tham biến
- **super**: dùng để tham chiếu đến instance của lớp cha gần nhất qua đó có thể truy cập thuộc tính và phương thức của đối tượng cha.
- **@Override**: sử dụng trong trường hợp định nghĩa lại các phương thức cho phù hợp với lớp con.
- **final**:
  - class: không cho phép tạo lớp con dẫn xuất từ lớp này.
  - method: không cho phép lớp con override phương thức của lớp cha.

1. Kế thừa trong Java

Hàm tạo trong kế thừa

- Hàm tạo không được thừa kế
- Hàm tạo của lớp con phải gọi tới hàm tạo của lớp cha thông qua **super(<ds tham số>)**
- Lời gọi tới hàm tạo khác phải là câu lệnh đầu tiên trong hàm tạo

```
public class Student extends Person{
    public String studentId;

    public Student(String name, String address, String
    studentId) {
        super(name, address);
        this.studentId = studentId;
    }
}
```

- Trong trường hợp lớp cha có hàm tạo mặc định thì không cần gọi một cách tường minh

1. Kế thừa trong Java – Ví dụ

1. Xây dựng lớp điểm 2D với các phương thức:

- Hiển thị tọa độ điểm
- Tính khoảng cách từ điểm đến gốc tọa độ

2. Xây dựng lớp điểm 3D kế thừa lớp điểm 2D với phương thức:

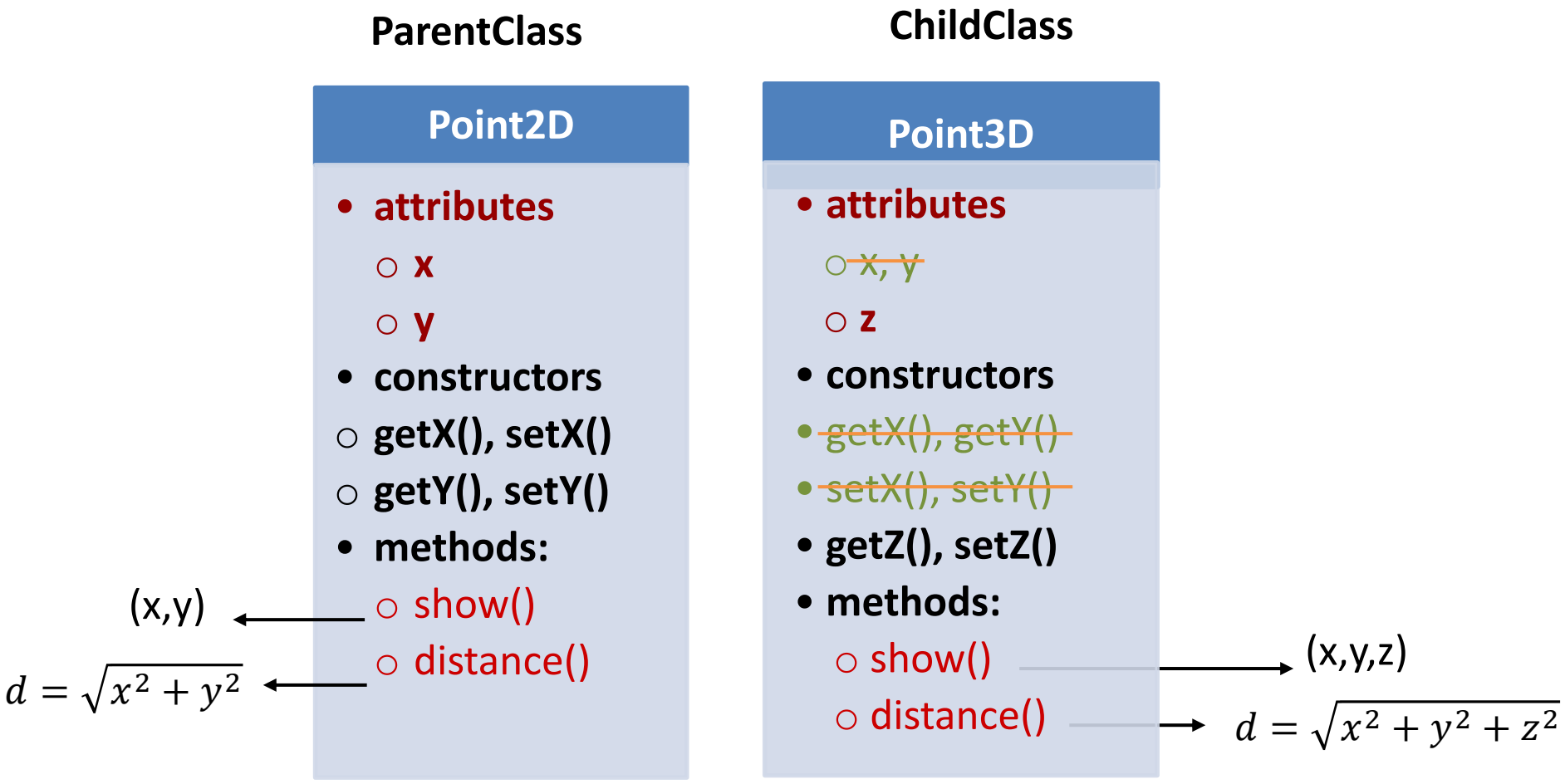
- Hiển thị tọa độ điểm
- Tính khoảng cách từ điểm đến gốc tọa độ

3. Xây dựng chương trình chính:

- Tạo danh sách điểm 2D và danh sách điểm 3D
- Tính tổng khoảng cách các điểm 2D đến gốc tọa độ
- Tính tổng khoảng cách các điểm 3D đến gốc tọa độ

1. Kế thừa trong Java

Xác định yêu cầu các lớp



1. Kế thừa trong Java – Ví dụ

```
public class Point2D {
    private double x,y;
    public Point2D(double x, double y){
        this.x=x;
        this.y=y;
    }
    public double getX(){
        return x;
    }
    public void setX(double x){
        this.x=x;
    }
    public double getY(){
        return y;
    }
    public void setY(double y){
        this.y=y;
    }
    public String show(){
        return "Point2D ["+this.x+","+this.y+"]";
    }
    public double distance(){
        return Math.sqrt(this.x*this.x+this.y*this.y);
    }
}
```

attributes

constructor

getters and setters

methods

1. Kế thừa trong Java – Ví dụ

```
public class Point3D extends Point2D{
    private double z;
    public Point3D(double x, double y, double z){
        super(x,y);
        this.z=z;
    }
    public double getZ(){
        return z;
    }
    public void setZ(double z){
        this.z=z;
    }
    @Override
    public String show(){
    return "Point3D [" + this.getX()+ "," + this.getY()+ "," + this.z
+"]";
    }
    @Override
    public double distance(){
        return Math.sqrt(super.distance() * super.distance() + this.z
* this.z);
    }
}
```

Thêm thuộc tính mới

Định nghĩa lại hàm tạo cho phù hợp với lớp con

thêm getters and setters cho thuộc tính mới của lớp con

định nghĩa lại methods

1. Kế thừa trong Java – Ví dụ

```
public static void main(String[] args) {
        List<Point2D> listPoint2D=new ArrayList<Point2D>();
        Point2D p1_2D = new Point2D(1.0,4.0);
        Point2D p2_2D = new Point2D(3.0,6.0);
        Point2D p3_2D = new Point2D(4.0,7.0);
//add elements to the list
        listPoint2D.add(p1_2D);
        listPoint2D.add(p2_2D);
        listPoint2D.add(p3_2D);
        for(Point2D point:listPoint2D){
                System.out.println(point.show());
        }
//distance calculation
        double sum2D=0.0;
        for(Point2D point:listPoint2D){
                sum2D +=point.distance();
        }
        System.out.println("Distance of Point2D: " +sum2D);
    }
```

Kết quả:

```
Point2D [1.0,4.0]
Point2D [3.0,6.0]
Point2D [4.0,7.0]
Distance of Point2D: 18.89356730641558
```

1. Kế thừa trong Java – Ví dụ

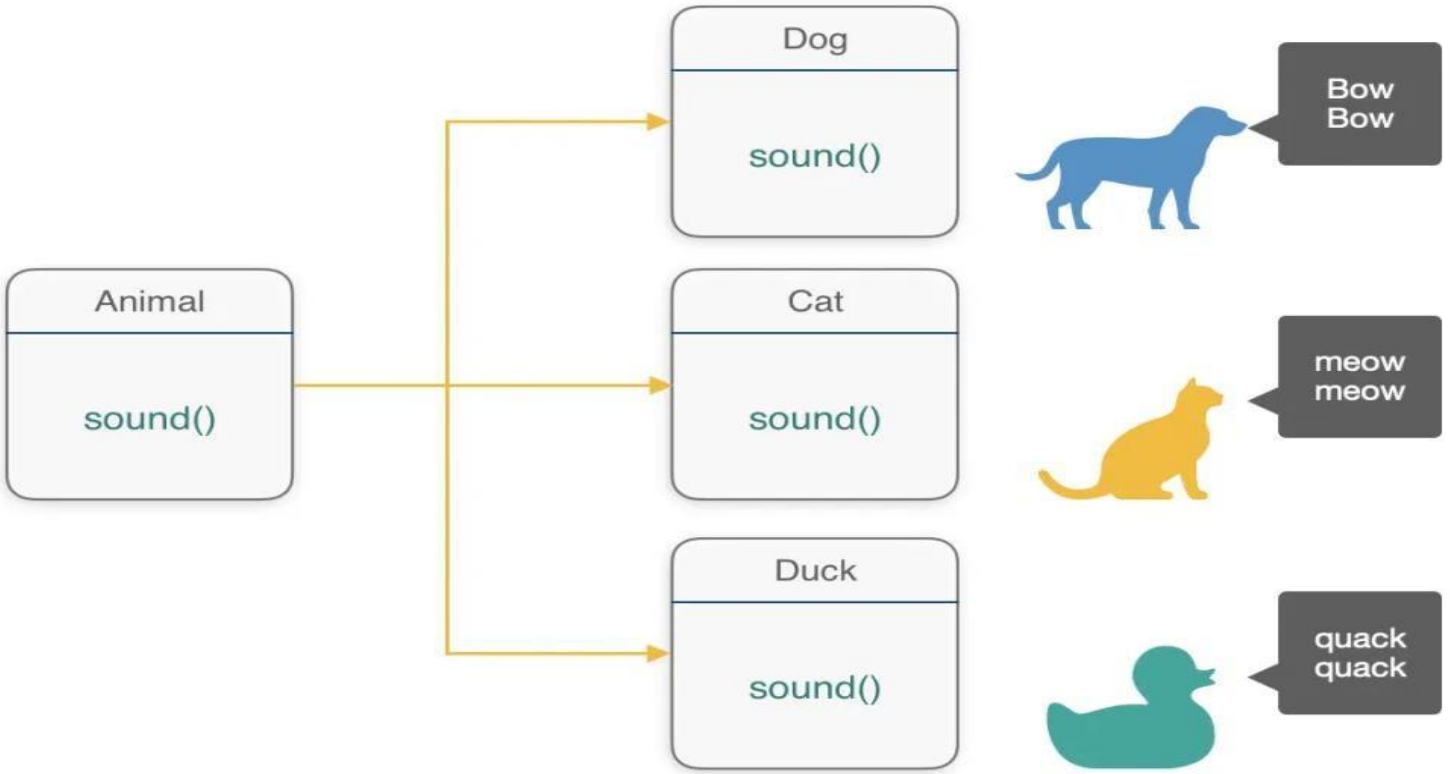
```
List<Point3D> listPoint3D=new ArrayList<Point3D >();
        Point3D p1_3D = new Point3D(1.0,4.0,7.0);
        Point3D p2_3D = new Point3D(3.0,6.0,9.0);
        Point3D p3_3D = new Point3D(5.0,7.0,3.0);
     //add elements to the list
        listPoint3D.add(p1_3D);
        listPoint3D.add(p2_3D);
        listPoint3D.add(p3_3D);
        for(Point3D point:listPoint3D){
            System.out.println(point.show());
        }
    //distance calculation
        double sum3D=0.0;
        for(Point3D point:listPoint3D){
            sum3D +=point.distance();
        }
        System.out.println("Distance of Point3D: " + sum3D);
```

Kết quả:

Point3D [1.0,4.0,7.0]
Point3D [3.0,6.0,9.0]
Point3D [5.0,7.0,3.0]
Distance of Point3D: 28.459444144102086

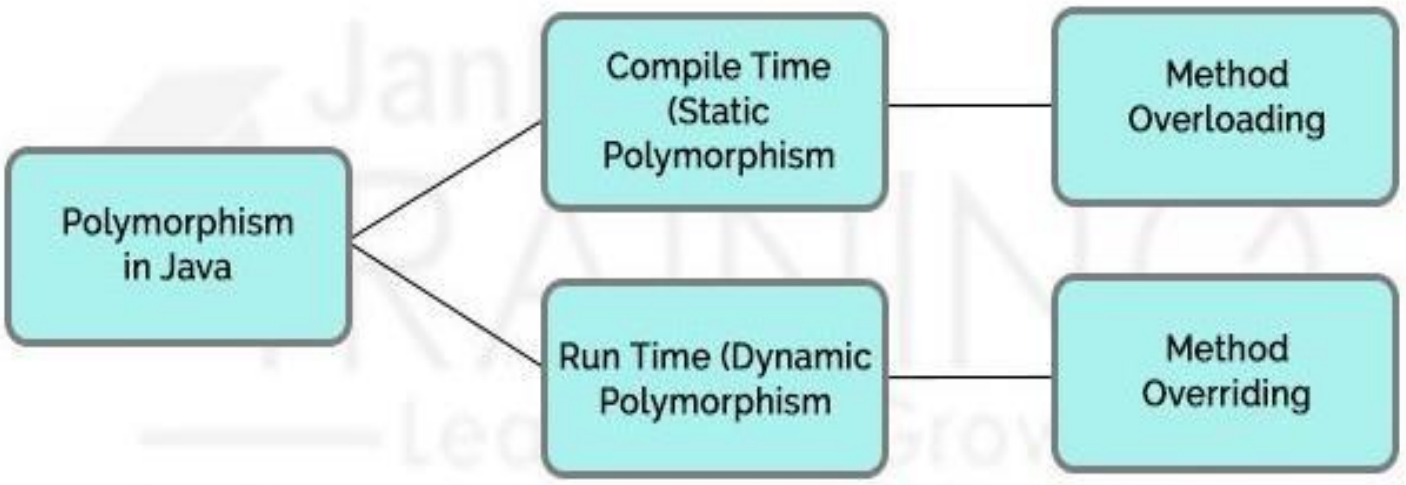
2. Đa hình trong Java

- Tính đa hình (polymorphism) là một trong bốn tính chất cơ bản của lập trình OOP trong Java, cho phép các đối tượng khác nhau thực thi chức năng giống nhau theo những cách khác nhau.



2. Đa hình trong Java

- Có hai kiểu của đa hình trong java, đó là đa hình lúc biên dịch (compile) và đa hình lúc thực thi (runtime). Chúng ta có thể thực hiện đa hình trong java bằng cách nạp chồng phương thức và ghi đè phương thức.

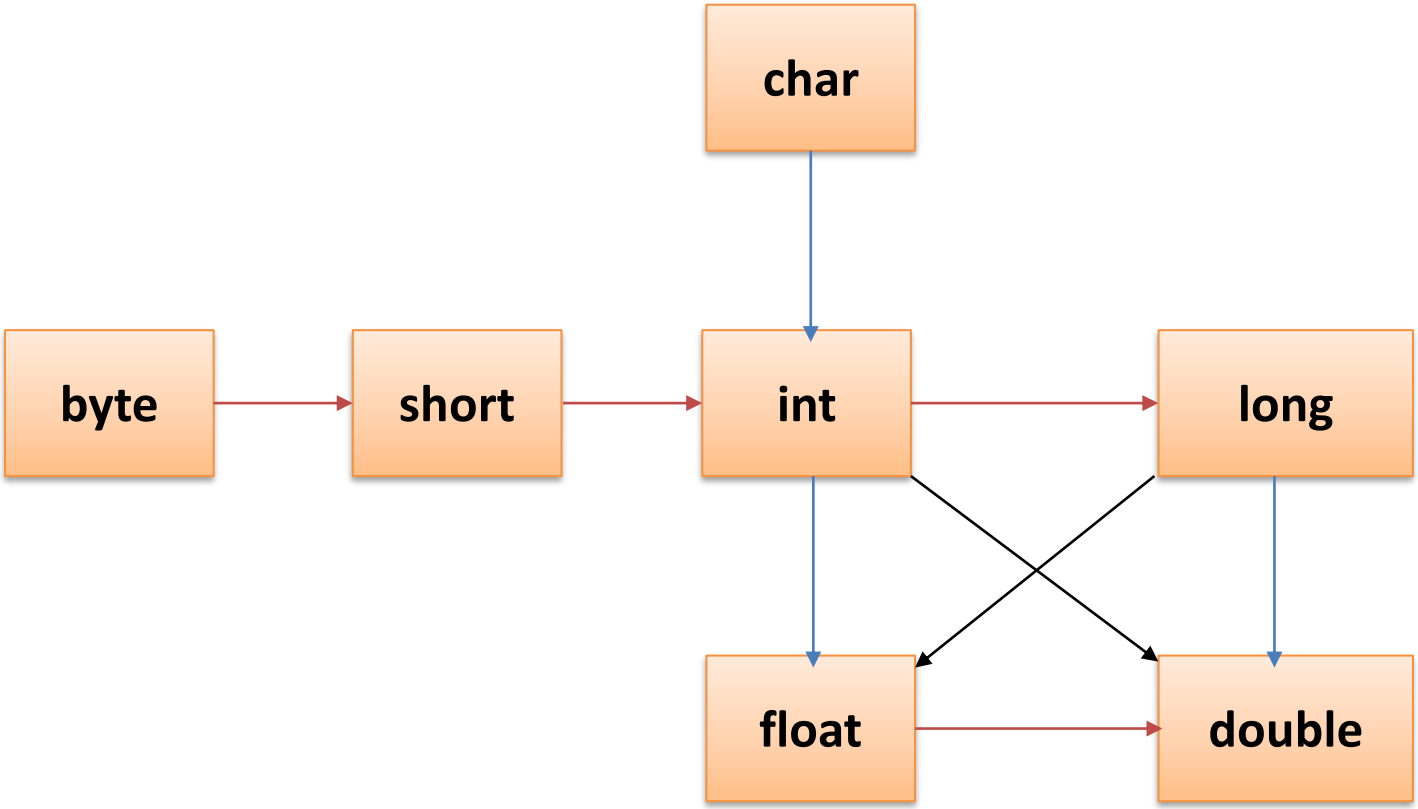


2. Đa hình trong Java

Method Overriding	Method Overloading
Khi lớp con thừa kế phương thức của lớp cha và cần định nghĩa lại cho phù hợp	Khi một lớp có nhiều phương thức cùng tên thực hiện mục đích giống nhau
Phương thức ghi đè cùng tên, cùng kiểu dữ liệu trả về, cùng tham số như nhau	Các phương thức phải khác kiểu tham số hoặc số lượng tham số
Phạm vi modifier của phương thức trong lớp con phải nằm trong phạm vi modifier của lớp cha	*Không thể nạp chồng phương thức bằng cách chỉ thay đổi kiểu trả về của phương thức bởi vì không biết phương thức nào sẽ được gọi*
Không ghi đè phương thức static	Java tự động ép kiểu nếu giá trị của đối số truyền vào không phù hợp với kiểu dữ liệu của tham số đã được đinh nghĩa

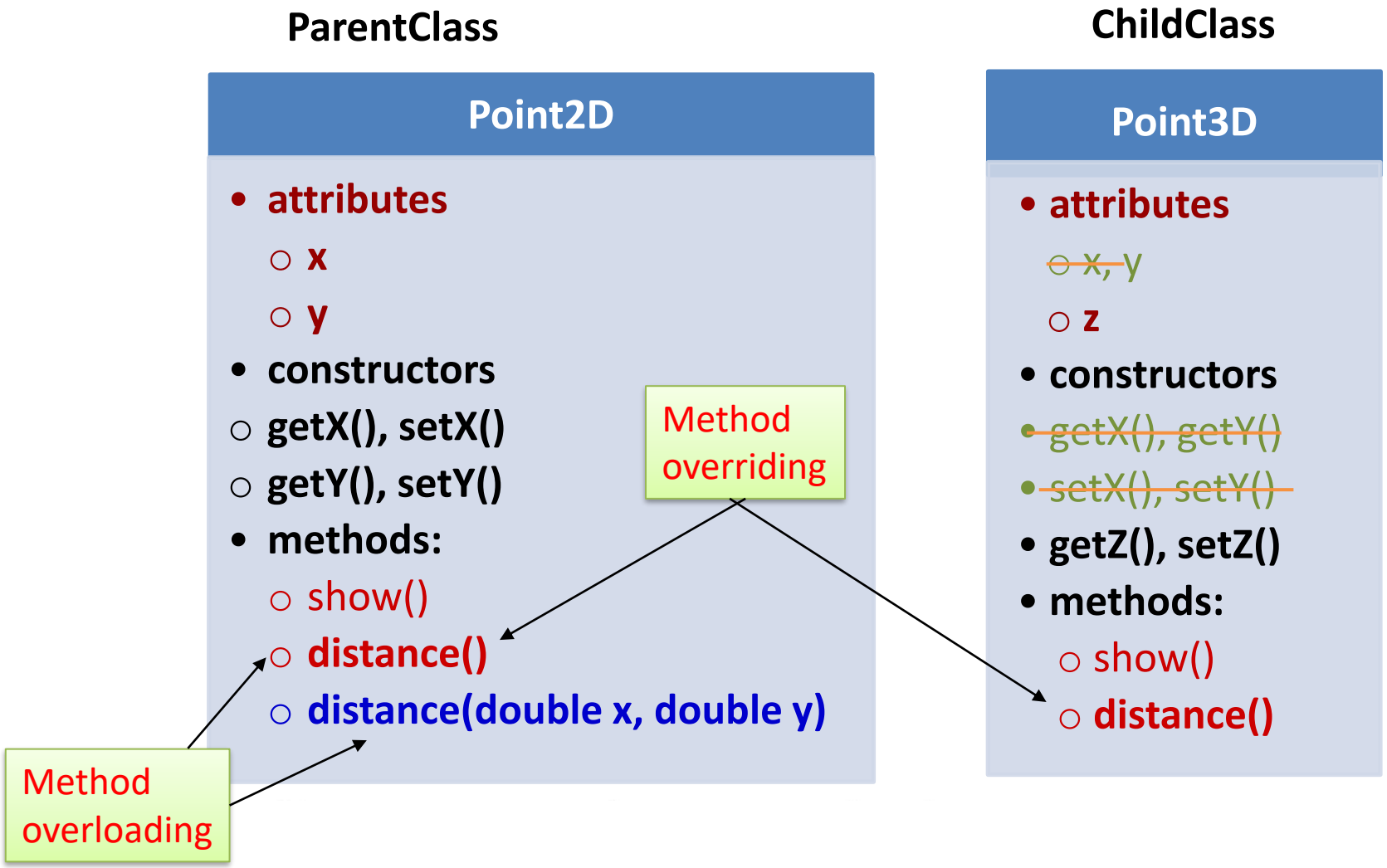
2. Đa hình trong Java

- ***Sơ đồ ép kiểu***



2. Đa hình trong Java – Ví dụ

Bổ sung thêm phương thức tính khoảng cách giữa hai điểm 2D



2. Đa hình trong Java – Ví dụ

```
public class Point2D {
public String show(){
        return "Point2D ["+this.x+","+this.y+"]";
    }
    public double distance(){
        return Math.sqrt(this.x*this.x+this.y*this.y);
    }
public double distance(double x, double y){
        double dx=(this.getX()-x);
        double dy=(this.getX()-y);
        return Math.sqrt(dx*dx+dy*dy);
    }
}
```

Method overloading

Method overriding

```
public class Point3D extends Point2D{
@Override
    public double distance(){
        return Math.sqrt(super.distance()* super.distance() +
this.z * this.z);
    }
}
```

2. Đa hình trong Java – Bài tập 1

Giả sử đã có các lớp:

- Khai báo lớp **TamGiac** với hàm tạo nhận vào 3 điểm. Viết phương thức tính chu vi và diện tích hình **TamGiac**.
- Khai báo lớp **ChuNhat** với hàm tạo nhận độ dài 2 cạnh. Viết phương thức tính chu vi và diện tính hình **ChuNhat**.
- Khai báo lớp **HinhTron** với hàm tạo nhận vào điểm O và bán kính. Viết phương thức tính chu vi và diện tích hình **HinhTron**.
- Viết chương trình chính khởi tạo 2 **TamGiac**, 2 **ChuNhat**, 2 **HinhTron** bất kỳ, in ra chu vi và diện tích các hình.

2. Đa hình trong Java – Bài tập 1

- Hãy khai báo lớp **Hinh** với 2 phương thức tính chu vi và diện tích
- Thay đổi các lớp **TamGiac**, **HinhTron**, **ChuNhat** để kế thừa lớp **Hinh** và ghi đè (override) phương thức tính chu vi và diện tích
- Thay đổi chương trình chính, lưu tất cả các hình vào 1 mảng **Hinh**.

2. Đa hình trong Java – Bài tập 1

Hãy cải tiến chương trình về hình cho phép người dùng nhập các hình từ bàn phím.

B1: Chương trình hỏi người dùng muốn nhập bao nhiêu hình?

B2: Chương trình hỏi người dùng muốn nhập vào hình gì? (1. Tam Giac, 2. Chu Nhat, 3. Hinh Tron)

B3: Tùy vào hình đã chọn chương trình hỏi người dùng:

Tam Giac: Nhập vào tọa độ 3 điểm

Chu Nhat: Nhập vào độ dài 2 cạnh

Hinh Tron: Nhập vào tâm O và bán kính

Sau khi nhập xong, chương trình in ra danh sách bao gồm: Loại hình, chu vi, diện tích

2. Đa hình trong Java – Bài tập 2

Xây dựng chương trình với các lớp đã cho để có thể lựa chọn và thực hiện các thao tác với từng lớp Sinh viên CNTT và Kinh tế.

Sinh viên

- **Mã, họ tên, ngày sinh, giới tính, tin đại cương, chính trị, pháp luật,…**
- **constructors**
- **methods:**
  - **Nhập thông tin**
  - **Hiển thị thông tin**

Sinh viên CNTT

- **CTDL_GT, Toán RR, CSDL, Lập trình OOP,..**
- **constructors**
- **methods:**
  - Nhập thông tin
  - Hiển thị thông tin
  - **Điểm TB**
  - **Xếp loại học tập**

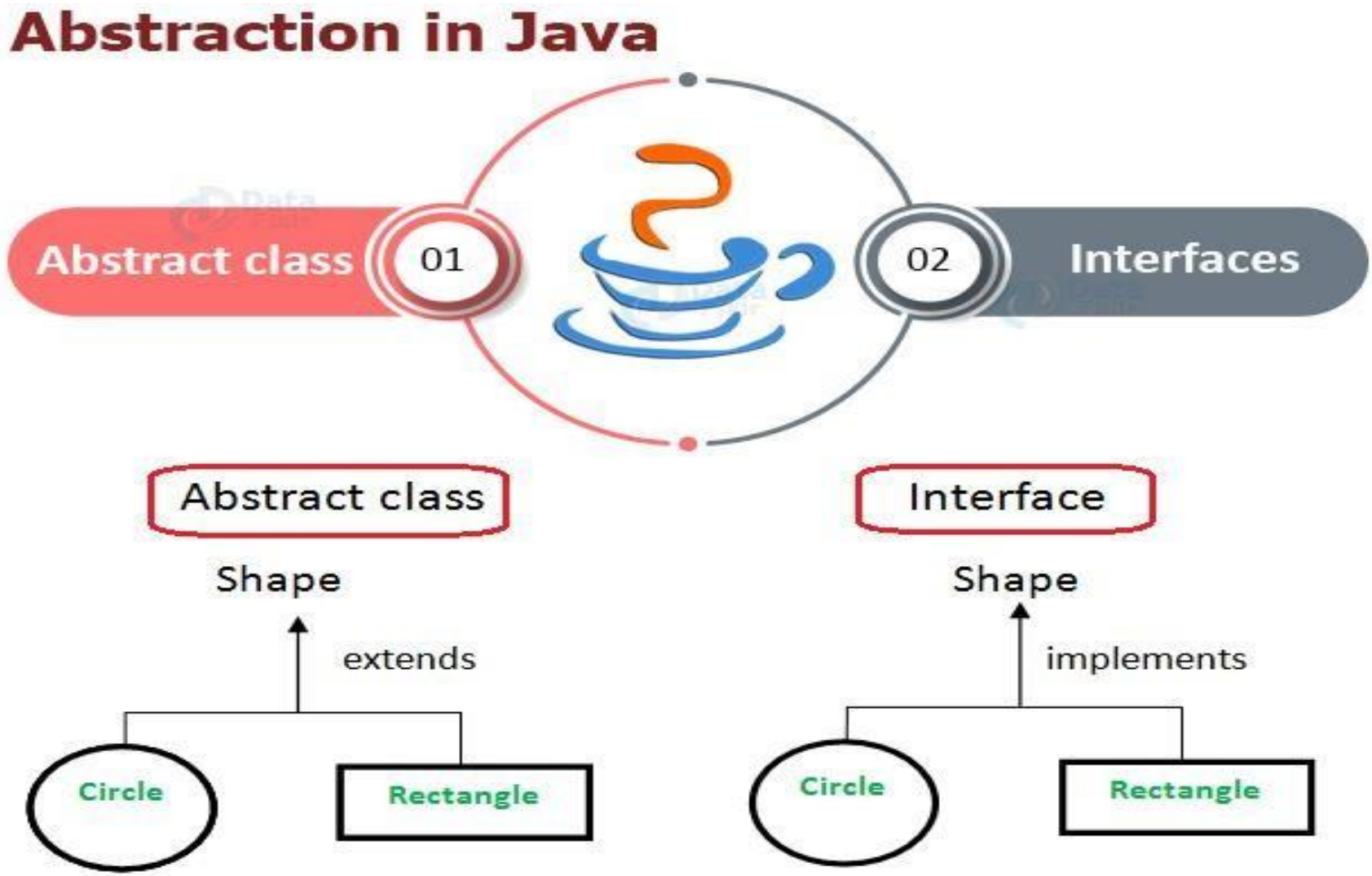
Sinh viên Kinh tế

- **Kinh tế vi mô, Kinh tế vĩ mô, Luật kinh tế, …**
- **constructors**
- **methods:**
  - Nhập thông tin
  - Hiển thị thông tin
  - **Điểm TB**
  - **Xếp loại học tập**

Chương 3. OOP – LỚP VÀ ĐỐI TƯỢNG TRONG JAVA

TÍNH TRỪU TƯỢNG

3. Tính trừu tượng trong Java



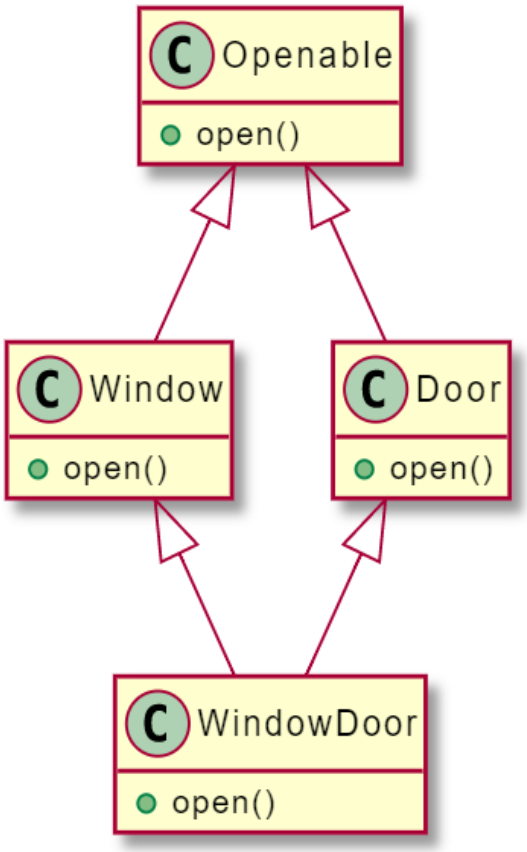
3. Tính trừu tượng trong Java

- **Mục đích:**
  - Cho phép ẩn đi các cài đặt chi tiết, chỉ hiển thị cách thức sử dụng đối với người dùng → giúp cho các lập trình viên cài đặt các nghiệp vụ phức tạp dựa trên các hành vi mà đối tượng được cung cấp.
  - Ví dụ:
    - Ứng dụng: các frameworks, các thư viện cung cấp cách tiếp cận trừu tượng cho phép chúng ta dựa trên đó để phát triển các ứng dụng riêng.
  - Trong Java, tiếp cận trừu tượng dựa trên kế thừa với:
    - Abstract class
    - Interface

3. Tính trừu tượng trong Java

Đa kế thừa – vấn đề gặp phải

- ***Diamond problem***: phương thức open() mà lớp con WindowDoor được kế thừa từ nhiều lớp cha → lựa chọn phương thức của lớp cha nào để kế thừa?
- Trong C++:
  - Yêu cầu lớp cha khai báo các phương thức open() là virtual (abstract) → lớp con phải định nghĩa lại.
  - Làm thế nào tạo instance cho lớp cha? → **pure virtual** vs **virtual**.
- Trong Java:
  - Nhóm các phương thức pure virtual → **interface**.
  - Các phương thức virtual được giữ nguyên → **abstract**.



3. Tính trừu tượng trong Java

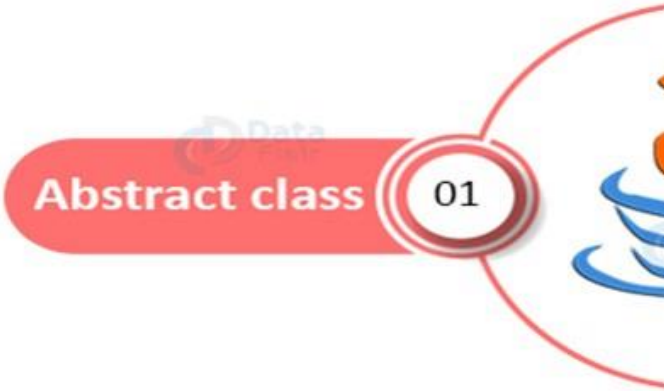
- **Abstract**

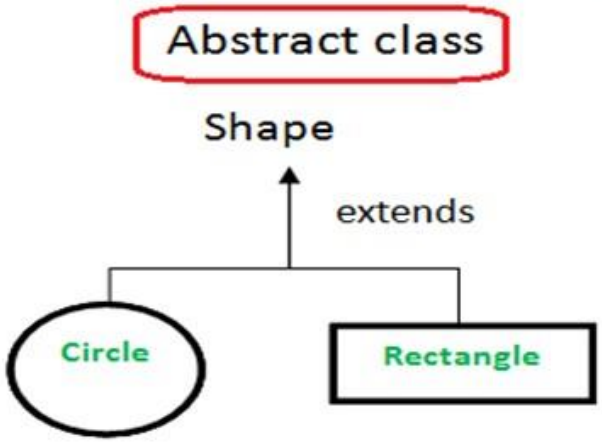
Là một class trong Java trong đó có chứa ít nhất một phương thức trừu tượng: phương thức này được định nghĩa prototype, phần cài đặt sẽ được thực hiện trong lớp dẫn xuất.

- Khai báo lớp trừu tượng thông qua từ khóa **abstract**
- Lớp trừu tượng là lớp có chứa phương thức trừu tượng

3. Tính trừu tượng trong Java

```
public abstract class Shape {
    abstract void draw();

    public static void main(String[] args) {
        Rectangle r = new Rectangle();
        Circle c = new Circle();
        r.draw();
        c.draw();
    }

    // Example - abstract [use the inner class]
    public static class Circle extends Shape {
        void draw() {
            System.out.println("I'm circle!");
        }
    }

    public static class Rectangle extends Shape {
        void draw() {
            System.out.println("I'm rectangle!");
        }
    }
}
```





3. Tính trừu tượng trong Java

Abstract

- Lớp abstract là lớp trừu tượng nên không cho phép khởi tạo đối tượng từ lớp abstract.
- Lớp dẫn xuất của lớp abstract có thể:
  - Là một lớp abstract → chưa cần cài đặt các phương thức abstract.
  - Là một lớp thường → bắt buộc phải cài đặt các phương thức abstract của lớp abstract.
- Trong Java, chỉ có thể kế thừa một lớp → kế thừa một lớp thường hoặc một lớp abstract.
- Phạm vi truy cập của phương thức abstract không thể là private vì nó cần được định nghĩa lại trong lớp dẫn xuất.
- Từ Java 7, lớp abstract có thể có hoặc không có phương thức abstract nhưng lớp abstract không thể tạo ra thực thể dù không có phương thức abstract.

3. Tính trừu tượng trong Java

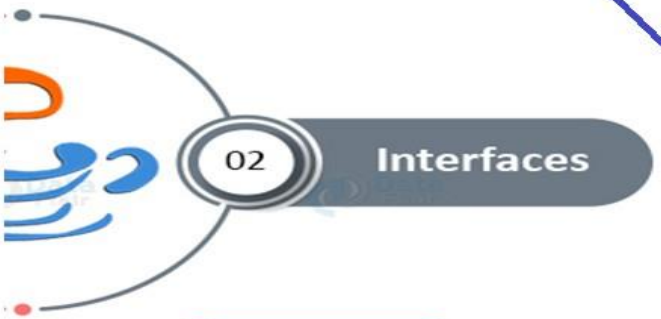
- **Interface**

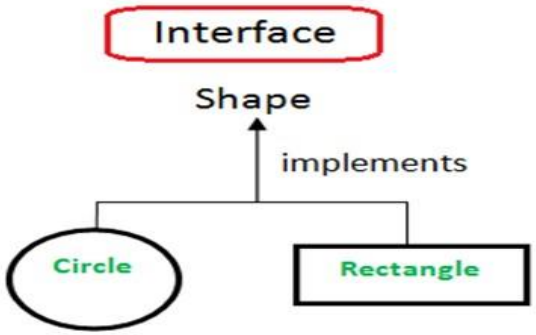
Là một cách thể hiện của đa kế thừa trong Java trong đó một **interface** định nghĩa prototype cho các phương thức – pure abstract: các phương thức này cài đặt sẽ được thực hiện trong lớp dẫn xuất [lớp cài đặt].

3. Tính trừu tượng trong Java

- **Interface**

```
package week_06;

public interface IShape {
    public void draw();
    public double area();
    public double perimeter();
}
```






```
public class Circle implements IShape {
    protected double x;
    protected double y;
    protected double r;

    public Circle(double x, double y, double r) {
        this.x = x;
        this.y = y;
        this.r = r;
    }

    public void draw() {
        System.out.println("I'm circle!");
    }

    public double area() {
        return this.r * this.r * Math.PI;
    }

    public double perimeter() {
        return this.r * 2.0 * Math.PI;
    }

    public static void main(String[] args) {
        Circle c = new Circle(2.0, 3.0, 5.0);
        c.draw();
        System.out.println("Area: " + c.area());
        System.out.println("Perimeter:" + c.perimeter());
    }
}
```

3. Tính trừu tượng trong Java

Interface

- Các phương thức được khai báo trong Interface đều là các phương thức pure abstract ➔ không cần dùng từ khóa abstract.
- Lớp cài đặt của interface có thể:
  - Là một lớp abstract.
  - Là một lớp thường ➔ bắt buộc phải cài đặt các phương thức **abstract** của interface.
- Trong Java, chỉ có thể kế thừa một lớp nhưng hỗ trợ đa cài đặt ➔kế thừa một lớp nhưng cài đặtnhiều interface.
- Phạm vi truy cập của phương thức trong interface không thể là private hoặc protected vì nó cần được cài đặt trong lớp cài đặt.
- Interface không phải là một lớp, chỉ là khai báo các phương thức cho họ các lớp ➔ không thể tạo ra thực thể từ interface.
- Trong interface không có khái niệm thuộc tính, chỉ có thể định nghĩa các biến/hàm toàn cục thông qua static/final.
- Một interface có thể kế thừa nhiềuinterface khác.

3. Tính trừu tượng trong Java

Abstract	Interface
Abstract class có phương thức **abstract** (không có thân hàm) hoặc phương thức **non-abstract** (có thân hàm).	Interface chỉ có phương thức **abstract**. Từ Java 8, nó có thêm **các phương thức default và static**.
Abstract class **không hỗ trợ đa kế thừa**.	Interface **có hỗ trợ đa kế thừa**
Abstract class có các biến **final, non-final, static and non-static**.	Interface chỉ có các biến **static và final**.
Abstract class **có thể cung cấp nội dung cài đặt cho phương thức của interface**.	Interface **không thể cung cấp nội dung cài đặt cho phương thức của abstract class**.
Từ khóa **abstract** được sử dụng để khai báo abstract class.	Từ khóa **interface** được sử dụng để khai báo interface.

Bài tập – 3

Interface – abstract class

Để xây dựng ứng dụng Paint cho phép người dùng thực hiện các thao tác với các hình bao gồm: Point, Circle, Line, Triangle, Rectangle, … người ta xây dựng không gian các hình với cách tiếp cận OOP.

- Xây dựng Interface Shape với các phương thức
  - Nhóm tính toán: diện tích (**area**), chu vi (**perimeter**), khoảng cách (**distance**),..
  - Nhóm biến đổi: tịnh tiến (**move**), xoay (**rotate**), thay đổi (**zoom**), …
- Dựa trên Interface Shape, xây dựng các lớp tương ứng với các đối tượng.

Bài tập – 3

- **Interface – abstract class**

Yêu cầu:

1. Cài đặt *<<interface>>* Shape dựa trên các mô tả yêu cầu
   - Định nghĩa các phương thức của Shape (tính toán + biến đổi).
2. Cài đặt các lớp tương ứng với các phương thức đã được mô tả trong *<<interface>>* Shape.
   - Point, Circle, Line, Triangle, Rectangle …
     - attributes/constructors/getters and setters/toString
     - area/perimeter/distance/move/rotate/zoom

Bài tập – 3

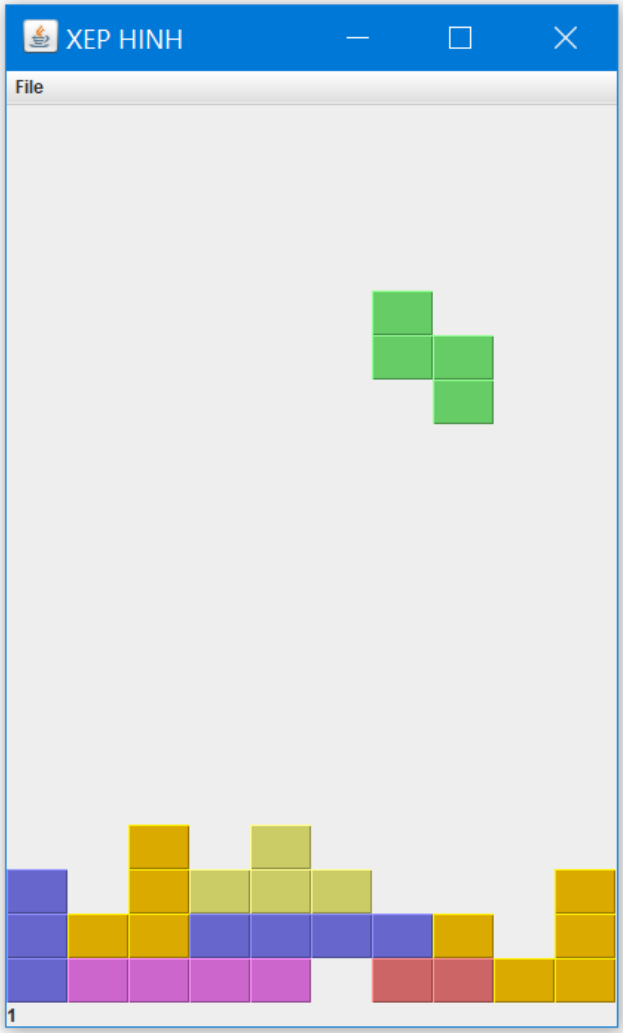
- **Interface – abstract class**

Yêu cầu:

3. Xây dựng chương trình chính:
   - Tạo một tập các hình ngẫu nhiên và đưa vào trong danh sách để quản lý.
   - Thực hiện hiển thị danh sách các hình đã tạo ra.
   - Tính tổng chu vi/diện tích các hình đã tạo ra và tìm hình có diện tích lớn nhất/ nhỏ nhất.
   - Biến đổi các hình bằng cách cho phép phóng to các hình với tỉ lệ ratio.
   - …
- Chú ý:
   - Sử dụng các khái niệm kế thừa, cũng như access modifier đối với các thuộc tính và phương thức.
   - Có thể xây dựng và sử dụng các abstract class nếu cần thiết.
   - Hiểu rõ cách thức khai báo và sử dụng hàm.

Bài tập - Xây dựng ứng dụng Game

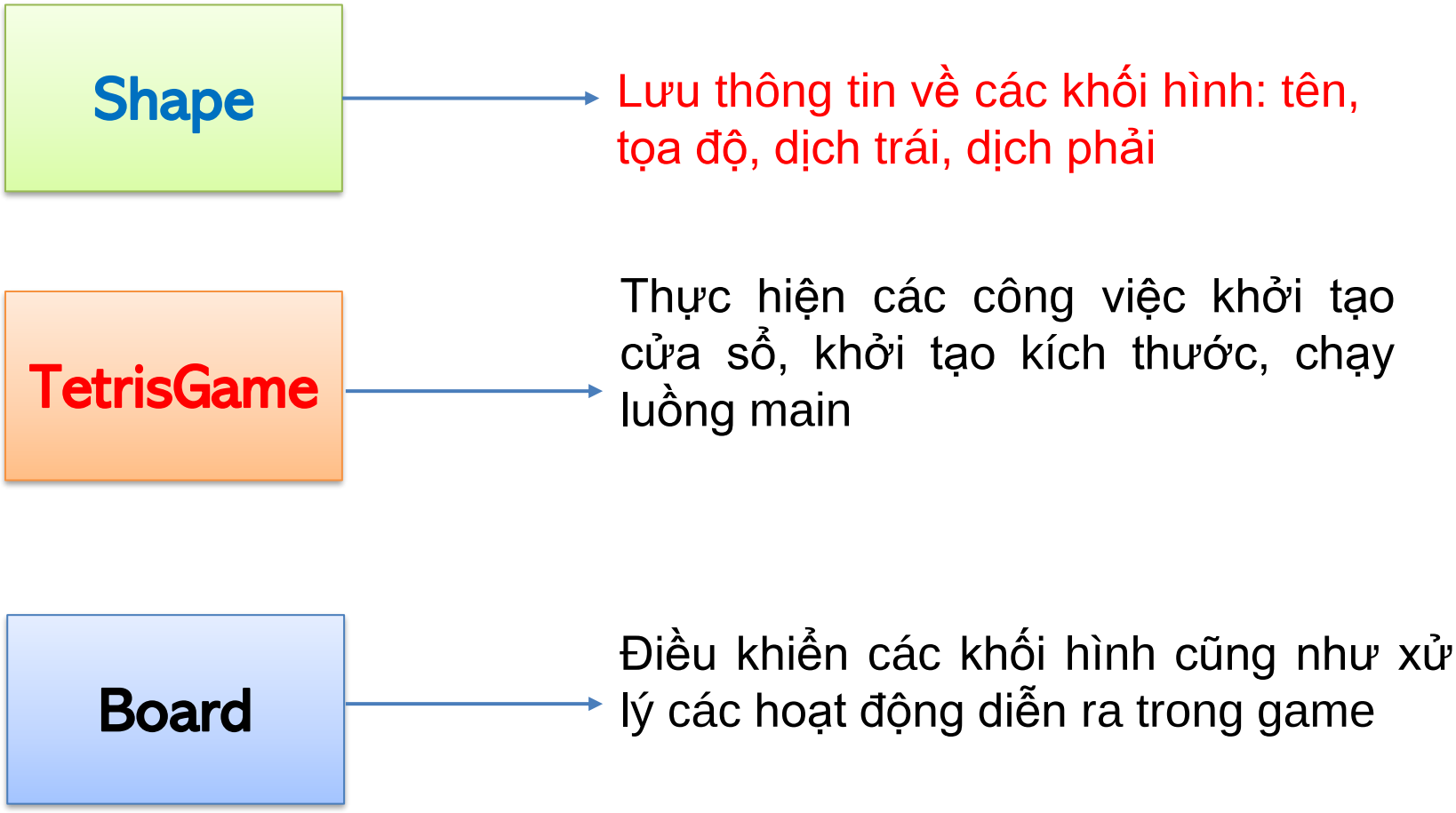
- Áp dụng xây dựng trò chơi xếp hình



Trong game có 7 khối gạch có dạng hình chữ **S,** hình chữ **Z**, hình chữ **T**, hình chữ **L**, hình **đường thẳng**, hình chữ **L ngược** và **hình vuông**. Mỗi khối hình được tạo nên bởi 4 hình vuông

Bài tập - Xây dựng ứng dụng Game

- Cần xây dựng **3 lớp** và sử dụng **các lớp có sẵn** trong Java



Bài tập - Xây dựng ứng dụng Game

❑ Lớp **Shape**

```
import java.util.Random;
public class Shape {
    protected enum Tetrominoes { NoShape,
ZShape, SShape, LineShape,  TShape,
SquareShape, LShape, MirroredLShape };
   private Tetrominoes pieceShape;
   private int coords[][];
   private int[][][] coordsTable;
   public Shape() {
      coords = new int[4][2];
      setShape(Tetrominoes.NoShape);
   }
    public void setShape(Tetrominoes shape) {
// tạo mảng coords để lưu giữ tọa độ thật của khối
hình.
```

```
coordsTable = new int[][][] {
        { { 0, 0 },  { 0, 0 },  { 0, 0 },  { 0, 0 } },
        { { 0,-1 },  { 0, 0 },  {-1, 0 },  {-1, 1 } },
        { { 0,-1 },  { 0, 0 },  { 1, 0 },  { 1, 1 } },
        { { 0,-1 },  { 0, 0 },  { 0, 1 },  { 0, 2 } },
        { {-1, 0 },  { 0, 0 },  { 1, 0 },  { 0, 1 } },
        { { 0, 0 },  { 1, 0 },  { 0, 1 },  { 1, 1 } },
        { {-1,-1 },  { 0,-1 },  { 0, 0 },  { 0, 1 } },
        { { 1,-1 },  { 0,-1 },  { 0, 0 },  { 0, 1 } }
      };
//Mảng coordsTable lưu tọa độ của các hình
vuông tạo nên từng loại khối hình

for  (int  i = 0; i < 4  ; i++) {

    for  (int  j = 0; j < 2; ++j) { coords[i][j] =
coordsTable[shape.ordinal()][i][j];
        }

     }    pieceShape = shape;
  }
```

Bài tập - Xây dựng ứng dụng Game

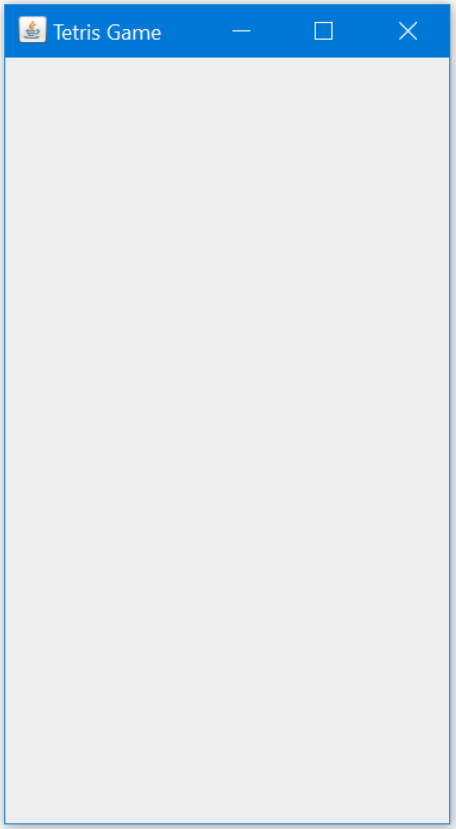
❑ Lớp **Shape**

```
public Shape rotateLeft() {
    if (pieceShape ==
Tetrominoes.SquareShape)
        return this;
    Shape result = new Shape();
    result.pieceShape = pieceShape;

    for (int i = 0; i < 4; ++i) {
        result.setX(i, y(i));
        result.setY(i, -x(i));
    }
    return result;
}
```

```
public Shape rotateRight() {
        if (pieceShape ==
Tetrominoes.SquareShape)
            return this;
        Shape result = new Shape();
        result.pieceShape = pieceShape;

        for (int i = 0; i < 4; ++i) {

            result.setX(i, -y(i));
            result.setY(i, x(i));
        }

        return result;
    }
}
```

Bài tập - Xây dựng ứng dụng Game

- Lớp **TetrisGame**

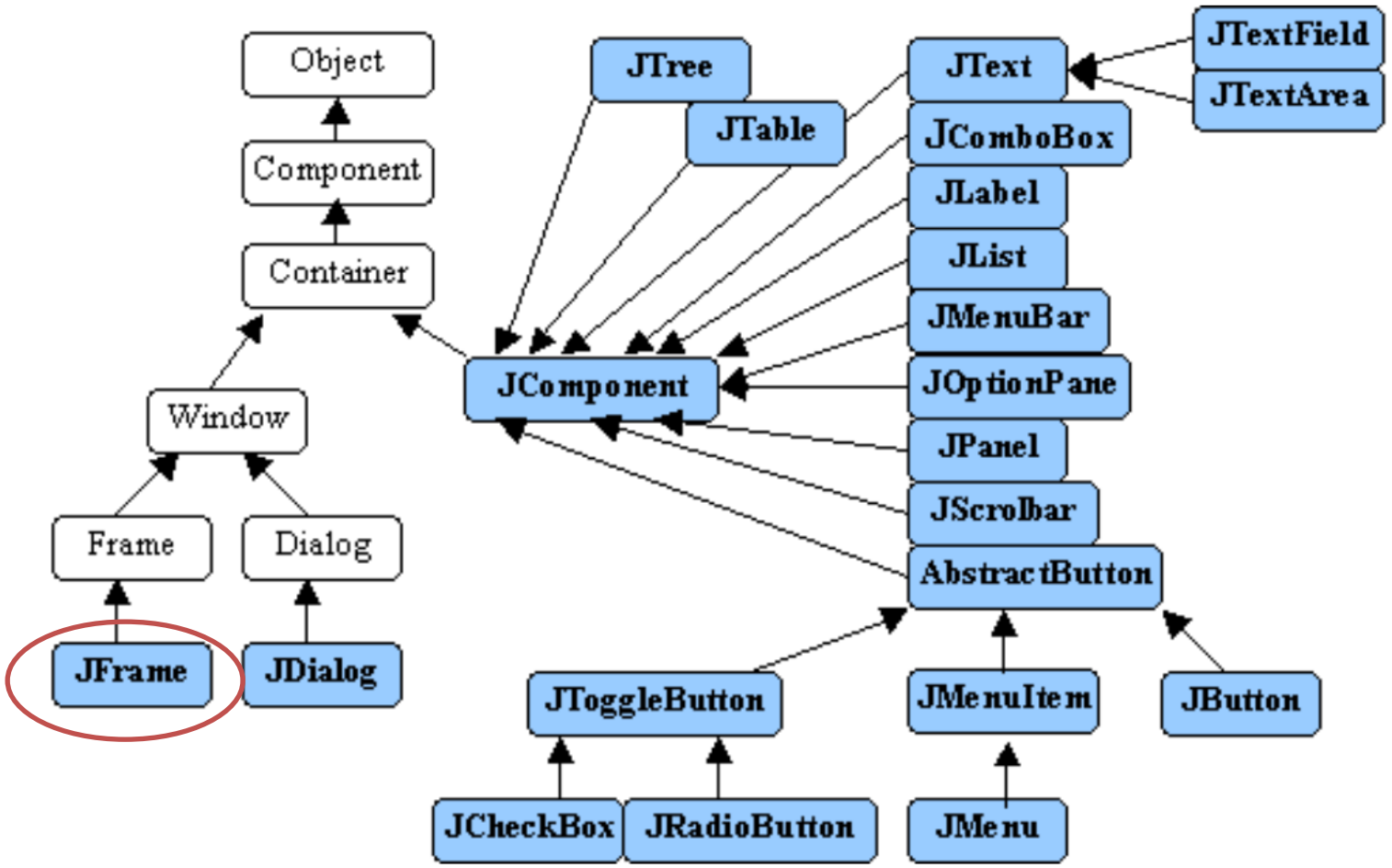


Để xây dựng lớp TetrisGame cần hiểu về thư viện các lớp chứa các thành phần hỗ trợ thiết kế giao diện

Bài tập - Xây dựng ứng dụng Game

- SWING là một thư viện chứa các thành phần để thiết kế giao diện. Các thành phần này là các lớp nằm trong gói **javax.swing.**

Kiến trúc SWING



Bài tập - Xây dựng ứng dụng Game

❑ Lớp TetrisGame

```
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
public class TetrisGame extends JFrame {
//Constructor
 public TetrisGame() {
            Board board = new Board(this);
            add(board);      //đưa đối tượng board vào frame
            board.start();   //gọi phương thức start() của lớp Board
            setSize(200, 400);   //thiết lập size cho frame
            setTitle("Tetri Game");   //thiết lập tiêu đề cho frame
   setDefaultCloseOperation(EXIT_ON_CLOSE); //thiết lập nút Close trên frame
   setLocationRelativeTo(null);  // hiển thị frame giữa màn hình.
  }
```

Bài tập - Xây dựng ứng dụng Game

- Lớp **TetrisGame**

```
public static void main(String[] args) {
    {
        @Override
        public void run() {
            TetrisGame game = new TetrisGame();
            game.setVisible(true); //display frame
        }
    });
  }
}
```

Bài tập - Xây dựng ứng dụng Game

❑ Lớp **Board**

- Khởi tạo Board
- Thiết lập thời gian di chuyển của khối hình
- Vẽ khối hình, tô màu
- Bắt sự kiện các phím di chuyển khi chơi

❑ Hoàn thiện Game

- Thêm menu hướng dẫn người chơi
- Thông báo trên giao diện như Start, OverGame,…
- Sử dụng phím tắt để chơi, dừng hoặc thoát Game

Bài tập-1

Sử dụng các khái niệm về kế thừa, đa hình và phạm vi truy cập đối với thuộc tính và phương thức để hoàn thiện bài tập.

1. Hoàn thiện lớp điểm 2D với các phương thức sau:

- Hiển thị tọa độ điểm
- Tính khoảng cách từ điểm đến gốc tọa độ
- Tính khoảng cách giữa 2 điểm
- Xác định điểm đối xứng qua gốc tọa độ

2. Hoàn thiện lớp điểm 3D kế thừa lớp điểm 2D với phương thức:

- Hiển thị tọa độ điểm
- Định nghĩa lại phương thức tính khoảng cách từ điểm đến gốc tọa độ
- Tính khoảng cách giữa 2 điểm 3D
- Xác định điểm đối xứng qua gốc tọa độ

3. Xây dựng chương trình chính:

- Tạo danh sách các điểm 2D và danh sách các điểm 3D
- Tính tổng khoảng cách các điểm 2D, tổng khoảng cách các điểm 3D
- Đưa các điểm đối xứng vào danh sách và hiển thị danh sách các điểm

Bài tập-2

- Xây dựng lớp **Person** gồm các thuộc tính họ tên, năm sinh, quê quán và các phương thức nhập/xuất các thông tin trên.
- Xây dựng lớp **Student** gồm ngoài các thông tin như lớp **Person** còn có điểm trung bình và nhập/xuất thông tin cho sinh viên.
- Xây dựng lớp **Teacher** gồm ngoài các thông tin như lớp **Person** còn có thêm số giờ dạy và nhập/xuất thông tin cho giảng viên.
- Thực hiện các phương thức nhập xuất thông tin của lớp **Student** và lớp **Teacher**

Mở rộng bài tập trên như sau:

- ✓ Thêm phương thức để xếp loại học tập cho sinh viên
- ✓ Thêm thuộc tính lương cơ bản, phụ cấp cho giảng viên và phương thức tính thu nhập = lương + phụ cấp
- ✓ Thực hiện nhập và hiển thị thông tin cho n sinh viên và n giảng viên

Bài tập - Xây dựng ứng dụng Game

1. Hoàn thiện lớp Shape và TetrisGame
2. Tìm hiểu cách xây dựng các xử lý cho lớp Board
   - Khởi tạo Board
   - Thiết lập thời gian di chuyển của khối hình
   - Vẽ khối hình, tô màu
   - Bắt sự kiện các phím di chuyển khi chơi

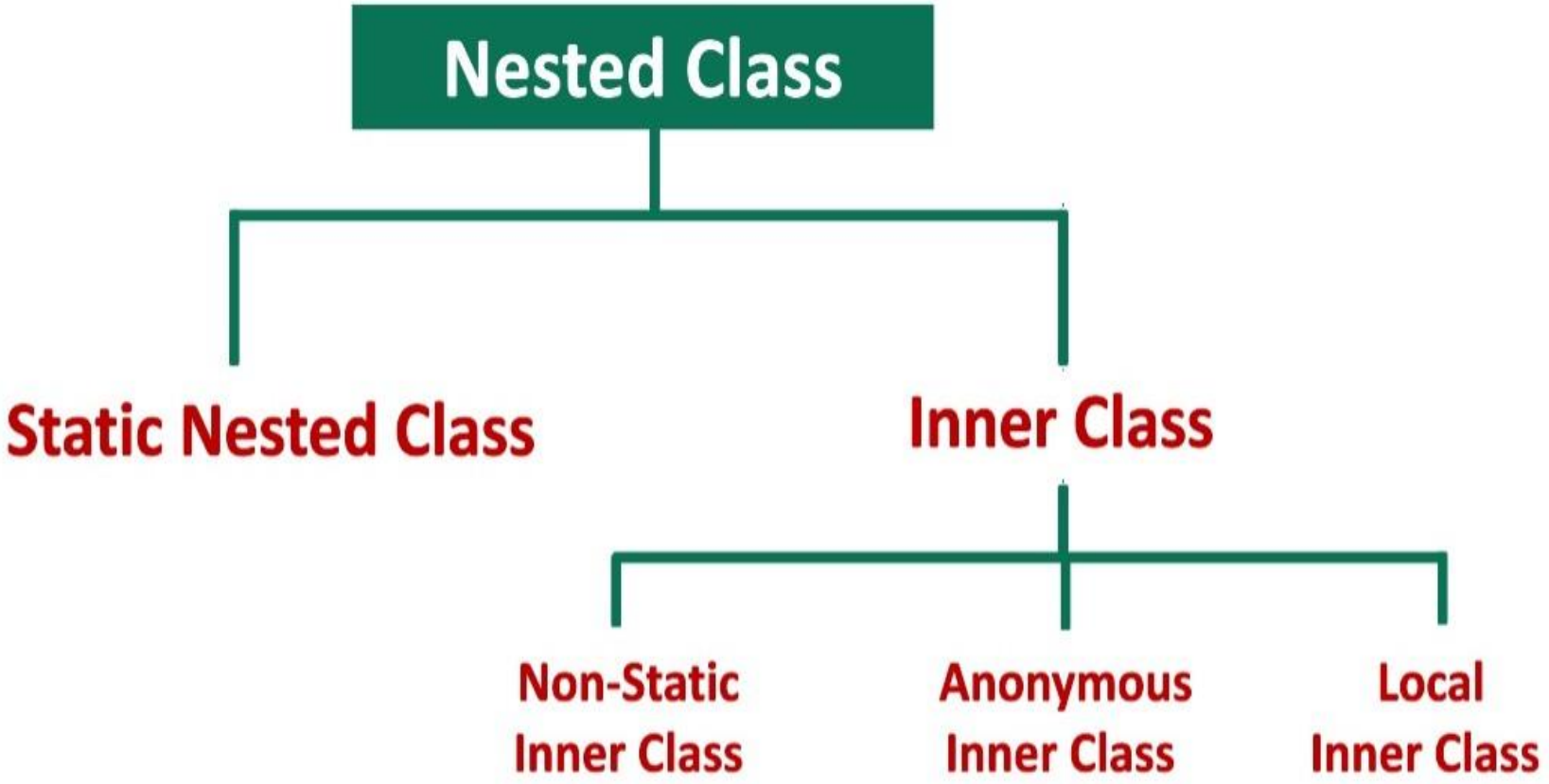
Chương 3. OOP – LỚP VÀ ĐỐI TƯỢNG TRONG JAVA

CÁC KHÁI NIỆM MỞ RỘNG

Inner/Outer class

Khái niệm:

- Trong Java, cho phép khai báo class/interface nằm trong một class/interface người ta gọi là lớp lồng nhau → nested class.
- Lớp chứa được gọi là outer class còn lớp khai báo bên trong được gọi là inner class.
- Ưu điểm:
  - Nhóm các lớp có cùng logic trong một tệp tin để quản lý.
  - Lớp inner khi đó được coi là một phần của lớp outer nên có thể truy cập đến các thuộc tính/phương thức của lớp outer ngay cả khi các thuộc tính/phương thức là private.
  - Tiết kiệm code trong việc truy xuất đến các thành phần của outer.
- Nhược điểm:
  - Không sử dụng được lớp inner, các phương thức của nó trong các lớp khác.
  - Inner class tham chiếu đến outer class nên đối tượng tạo ra chỉ được giải phóng khi đối tượng outer được giải phóng lãng phí bộ nhớ. Không nên dùng các mảng inner.


Nested Class
Static Nested Class
Inner Class
Non-Static
Inner Class
Anonymous
Inner Class
Local
Inner Class

Inner/Outer class

- **Static nested class**
  - Được coi như một thành viên static của lớp bên ngoài →
    - Có thể được truy cập thông qua lớp bên ngoài.
    - Chỉ truy cập vào các thành phần static của lớp bên ngoài.

```
public class StaticNestedClass {
    private static String data = "I'm static data!";

    // Static inner class
    public static class InnerClass{
        public void show(){
            System.out.println(data);
        }
    }

    public static void main(String[] args) {
        StaticNestedClass.InnerClass s = new StaticNestedClass.InnerClass();
        s.show();
    }
}
```

Inner/Outer class

- **Non static nested class – inner class**

  Được coi như là 2 lớp khi biên dịch tuy nhiên

  - Được truy cập thông qua lớp bên ngoài.
  - Có thể truy cập vào các thành phần thuộc tính/phương thức của lớp bên ngoài.

```
public class NonStaticNestedClass {

    private int data = 30;

    public void showOuter(){
        System.out.println("I'm outer class!");
    }

    public class InnerClass {
        private void show() {
            showOuter();
            System.out.println("Data is " + data);
        }
    }

    public static void main(String args[]) {
        NonStaticNestedClass outer = new NonStaticNestedClass();
        NonStaticNestedClass.InnerClass inner = outer.new InnerClass();
        inner.show();
    }
}
```

Inner/Outer class

- **Local inner class**
  - Lớp inner được khai báo trong một phương thức của lớp outer. Khi đó lớp inner sẽ không thể được gọi từ bên ngoài.

```
public class LocalInnerClass {

    private int data = 30;

    void showOuter(){
        System.out.println("I'm outer class");
    }

    void display() {
        class InnerClass {
            void show() {
                showOuter();
                System.out.println(data);
            }
        }
        InnerClass inner = new InnerClass();
        inner.show();
    }

    public static void main(String args[]) {
        LocalInnerClass obj = new LocalInnerClass();
        obj.display();
    }
}
```

Inner/Outer class

- **Anonymous inner class**
  - Lớp inner được tạo ra trong quá trình cài đặt các phương thức ảo của một abstract class hoặc interface.

```
package week_06;

public class AnonymousClass {
    public static void main(String[] args) {
        IShape s = new IShape() {
            private double r = 5;

            public void draw() {
                System.out.println("I'm anonymous class - circle.");
            }

            public double area() {
                return Math.PI * r * r;
            }

            public double perimeter() {
                return Math.PI * 2.0 * r;
            }
        };
        s.draw();
        System.out.println(s.area());
        System.out.println(s.perimeter());
    }
}
```