



CHƯƠNG 3 LỚP VÀ ĐỐI TƯỢNG

ThS. Phạm Văn Tiệp

Nội dung

- Đóng gói và xây dựng lớp
- Khai báo và sử dụng đối tượng
- Tái sử dụng mã nguồn
- Kết tập



ĐÓNG GÓI VÀ XÂY DỰNG LỚP

Đóng gói

- Lớp đóng gói các thành viên và chỉ định điều khiển truy cập tới các thành viên đó:
 - ✓ **Thuộc tính**
 - ✓ **Phương thức**
- Tập hợp các lớp được nhóm lại thành gói (package). Mỗi lớp trong gói cũng được chỉ định điều khiển truy cập
- Các từ khóa chỉ định điều khiển truy cập trong Java:
 - ✓ **public: có thể truy cập từ mọi nơi**
 - ✓ **protected: có thể truy cập từ trong gói hoặc từ các lớp con**
 - ✓ **Không chỉ định: có thể truy cập từ trong gói**
 - ✓ **private: chỉ có thể truy cập từ chính lớp đó**

Các bước xây dựng lớp

- **Bước 1:** Mô hình hóa lớp đối tượng
Phát hiện các thuộc tính và hành vi
- **Bước 2:** Mô tả phần tiêu đề của lớp
- **Bước 3:** Định nghĩa thuộc tính
- **Bước 4:** Định nghĩa phương thức khởi tạo (Constructor)
- **Bước 5:** Định nghĩa phương thức

Mô hình hoá lớp đối tượng

- Bước này thực hiện trừu tượng hóa các đối tượng thực
- Định nghĩa một lớp cần có:
 - ✓ Tên lớp
 - ✓ Danh sách các thuộc tính
 - ✓ (Có thể cần) Phương thức khởi tạo
 - ✓ Danh sách các phương thức

Mô tả tiêu đề của lớp

- **Cú pháp:**

```
package pack.name;
```

```
/**Javadoc*/
```

```
Modifier class ClassName{
```

```
    //class's body
```

```
}
```

} Tiêu đề của lớp
(Class Header)

- **Trong đó:**

pack.name : tên gói

/Javadoc*/**: chú thích để tạo tài liệu Javadoc (không bắt buộc)

Modifier: chỉ định truy cập (chỉ có thể là public hoặc không chỉ định)

ClassName : tên lớp, theo quy tắc định danh của Java

Khái niệm gói (package)

- **Khái niệm gói:** Là một nhóm các lớp có liên hệ với nhau được tổ chức thành một đơn vị để quản lý.
- **Cách tổ chức lớp:** các gói được tổ chức phân cấp theo cây thư mục, mỗi gói tương ứng với một thư mục chứa các lớp trong đó.
- **Cách tạo gói cho các lớp**
 - ✓ Các lớp được tạo hoặc chép vào thư mục ứng với gói, trong đó mỗi tệp chương trình trong gói phải sử dụng từ câu lệnh sau để định nghĩa gói.

`package tên_gói_cấp1 . tên_gói_cấp2;`

Khái niệm gói (package)

- **Cách sử dụng gói và không gian tên của lớp**
 - ✓ Để sử dụng gói trong một chương trình ta sử dụng lệnh **import**, khi đó chương trình có thể dùng tất cả các lớp của gói đó.
 - ✓ Có thể sử dụng các lớp trực tiếp với tên đầy đủ như sau:
tên_gói_cấp1 · tên_gói_cấp2 · ... tên_lớp

Khái báo thuộc tính

- **Cú pháp:**

```
/**Javadoc*/  
Modifier DataType attributeName;
```

- **Trong đó:**

/Javadoc*/**: chú thích để tạo tài liệu Javadoc (không bắt buộc)

Modifier: chỉ định truy cập

Datatype: kiểu dữ liệu, có thể là một lớp lồng (nested class)

attributeName: tên thuộc tính, theo quy tắc định danh

Khai báo thành viên hằng số

- **Cú pháp:**

```
Modifier final DataType CONST_NAME = value;
```

- **Trong đó:**

Modifier: chỉ định truy cập

Datatype: kiểu dữ liệu

CONST_NAME: tên hằng

value: giá trị gán cho hằng

- **Ví dụ**

```
public final int MAX_STUDENT = 100;
```

Khai báo phương thức

- **Cú pháp:**

```
/**Javadoc*/  
Modifier DataType methodName(parameterList){  
    // method's body  
}
```

- **Trong đó:**

/Javadoc*/**: chú thích để tạo tài liệu Javadoc (không bắt buộc)

Modifier : chỉ định truy cập

Datatype: kiểu dữ liệu trả về của phương thức

methodName: tên phương thức

parameterList: danh sách các tham số, bao gồm kiểu dữ liệu và tên của tham số

Lệnh return

- Nếu **DataType** là **void**: phương thức không trả về giá trị
- Ngược lại, trong nội dung của phương thức cần sử dụng lệnh **return** để trả về giá trị có kiểu dữ liệu tương ứng
- Khi câu lệnh **return** được thực hiện, phương thức sẽ kết thúc
- **Lưu ý:**
 - ✓ Việc sử dụng **return** nhiều lần trong 1 phương thức có thể khiến chương trình mất cấu trúc
 - ✓ Nên sử dụng 1 biến cục bộ lưu và tính giá trị trả về trước khi sử dụng **return**
 - ✓ Cố gắng 1 phương thức chỉ có một câu lệnh **return**

Phương thức khởi tạo

- Khi một đối tượng tạo ra, một phương thức đặc biệt được gọi là **phương thức khởi tạo** được tự động gọi:
 - ✓ Gán giá trị ban đầu cho các thuộc tính
 - ✓ Có thể thực hiện một số thao tác xử lý
 - ✓ Một lớp có thể có nhiều phương thức khởi tạo
- **Cú pháp:**

```
/**Javadoc*/  
Modifier ClassName(parameterList){  
    // method's body  
}
```

Phương thức khởi tạo

- Phương thức khởi tạo mặc định (nên có)

```
/**Javadoc*/  
Modifier ClassName(){  
    // method's body  
}
```

Thành viên lớp

- Khi định nghĩa lớp, ta có thể khai báo một số thành viên với tư cách là thành viên của lớp:
 - ✓ **Thành viên lớp:** các thành viên có thể được truy cập mà không cần qua một đối tượng (thể hiện của lớp)
- **Cách thức:** khai báo thành viên với từ khóa **static**

```
Modifier static DataType attributeName;
```

```
Modifier static DataType methodName(parameterList){  
    // method's body  
}
```

```
Modifier static final DataType CONST_NAME = value;
```

Thành viên lớp

Thành viên đối tượng	Thành viên lớp
<ul style="list-style-type: none">Chỉ được truy cập thông qua một đối tượng (thể hiện của lớp)Thuộc tính của các đối tượng có giá trị khác nhauSự thay đổi giá trị thuộc tính của một đối tượng này là độc lập với các đối tượng khác	<ul style="list-style-type: none">Có thể truy cập thông qua tên lớp hoặc qua đối tượngThuộc tính của đối tượng khác nhau có giá trị giống nhauSự thay đổi giá trị thuộc tính của đối tượng này kéo theo sự thay đổi tương ứng trên các đối tượng khác

Thành viên lớp

- Thay đổi giá trị một **thuộc tính static** trong một đối tượng này sẽ thay đổi giá trị thuộc tính đó của các đối tượng khác cùng lớp
- Một **phương thức static** chỉ có thể truy cập vào **các thuộc tính static** và chỉ có thể gọi **các phương thức static** cùng lớp

Bảo vệ dữ liệu của đối tượng

- Thuộc tính của lớp thường được đặt chỉ định truy cập là **private**
- Truy cập vào thuộc tính của đối tượng thông qua các hàm **setter** (thay đổi giá trị của thuộc tính) và hàm **getter** (lấy giá trị của thuộc tính)
- **Mục đích:** **che giấu** và **kiểm soát** giá trị truyền cho thuộc tính

```
private int intAttr;  
  
//Setter method  
public void setIntAttr(int intAttr){ this.intAttr = intAttr; }  
  
//Getter method  
public int getIntAttr (){ return this.intAttr; }
```

Từ khóa **this** cho phép tự tham chiếu đến chính đối tượng đó

Ví dụ về xây dựng lớp

- Tạo một đối tượng “học phần” có
 - ✓ Mã môn, tên môn, số lượng đăng ký tối đa và số lượng đăng ký hiện tại
 - ✓ Cho phép người dùng đăng ký một SV vào lớp, hủy đăng ký một SV và hiển thị thông tin môn học
- **Yêu cầu:**
 - ✓ Không thể đăng ký nữa nếu số đăng ký đã bằng định mức
 - ✓ Không thể hủy đăng ký nếu không có sinh viên

Lớp Subject

- Phát hiện thuộc tính:
 - ✓ subjectID: Mã học phần –Xâu ký tự
 - ✓ subjectName: Tên học phần –Xâu ký tự
 - ✓ quota: Số lượng sinh viên tối đa được đăng ký
 - ✓ currentEnrolment: Số SV đăng ký hiện tại
- Phát hiện phương thức
 - ✓ enrolStudent(): Đăng ký một sinh viên
 - ✓ unEnrolStudent(): Hủy đăng ký
 - ✓ displaySubjectInfo(): Hiển thị thông tin đăng ký
 - ✓ Các phương thức setter và getter
 - ✓ Phương thức khởi tạo

Định nghĩa lớp Subject

```
package java.oop.subject

/** The Subject class illustrates a subject */

public class Subject {

    private String subjectID;
    private String subjectName;
    private int quota;
    private int currentEnrolment;

    /** The default constructor
    public Subject(){
        this.quota = 0;
    }

    //Declare other methods...

}
```

Constructor

/** Constructs a newly created Subject object by assigning initial values for attributes

* @param initID : the ID of subject

* @param initName : the name of subject

* @param initQuota : the quota of subject

* @param initCurrentEnrolment: the initial number of students enrolling

*/

```
public Subject(String initID, String initName, int initQuota, int initCurrentEnrolment){  
    this.subjectID = new String(initID);  
    this.subjectName = new String (initName);  
    this.quota = initQuota;  
    this.currentEnrolment = initCurrentEnrolment;  
}
```


enrolStudent()

/** Enrols a student into course*/

```
public void enrolStudent(){  
    System.out.print("Enrolling student... ");  
    if (currentEnrolment < quota){  
        ++currentEnrolment;  
        System.out.println("Student enrolled in " + subjectName);  
    }  
    else  
        System.out.println("Quota reached, enrolment failed");  
}
```

unEnrolStudent()

/** Cancels an enrolment*/

```
public void unEnrolStudent(){  
    System.out.print("Un-enrolling student... ");  
    if (currentEnrolment <= 0)  
        System.out.println("No students to un-enrol");  
    else{  
        --currentEnrolment;  
        System.out.println("Student un-enrolled from " + subjectName);  
    }  
}
```

displaySubjectInfo()

/** Displays the information of the course*/

```
public void displaySubjectInfo(){  
    System.out.println("Subject ID: " + subjectID);  
    System.out.println("Subject name: " + subjectName);  
    System.out.println("Quota: " + quota);  
    System.out.println("Currently enrolled: " + currentEnrolment);  
    int availablePlaces = quota - currentEnrolment;  
    System.out.println("Can accept " + availablePlaces +  
        " more students");  
}
```



KHAI BÁO VÀ SỬ DỤNG ĐỐI TƯỢNG

Khái báo và sử dụng đối tượng

```
package java.oop.example.static  
  
import java.oop.example.StaticAttribute  
  
/** The StaticAttributeTest class illustrates accessing static attribute */  
  
public class StaticAttributeTest {  
    public static void main (String[] args) {  
        StaticAttribute attr1 = new StaticAttribute ();  
        attr1.setStaticAttr(1);  
  
        StaticAttribute attr2 = new StaticAttribute ();  
        attr2.setStaticAttr(-1);  
  
        System.out.println("Static attribute in attr1: " + attr1.getStaticAttr());  
        System.out.println("Static attribute in attr2: " + attr2.getStaticAttr());  
    }  
}
```


Khai báo và sử dụng đối tượng

- **Cú pháp:**

```
ClassName objectName = new ClassName();
```

- **Trong đó:**

ClassName: Tên lớp mà đối tượng tạo ra từ lớp đó

objectName: Tên đối tượng

ClassName(): Phương thức khởi tạo. Tên phương thức khởi tạo trùng với tên lớp.

- Truy cập tới các thành viên của đối tượng qua toán tử ‘.’
- Lưu ý tới chỉ định điều khiển truy cập
- Các đối tượng được khai báo mà không khởi tạo sẽ mang giá trị null

Khái báo và sử dụng đối tượng

- Có thể tách thành 2 câu lệnh:

```
ClassName objectName;  
objectName = new ClassName();
```

- Điều gì xảy ra khi 2 câu lệnh này được thực thi:

ClassName objectName;

new ClassName();



Giá trị của **objectName** là địa chỉ vùng nhớ chứa thông tin của đối tượng trong bộ nhớ heap

Tạo và sử dụng Subject

```
package java.oop.subject;

/** The SubjectManagement class manages subjects*/

public class SubjectManagement {

    public static void main(String[] args) {

        Subject javaPrograming = new Subject("IT", "Java Programing", 40, 0);

        javaPrograming.unEnrolStudent();

        javaPrograming.displayInfor();

        for (int i = 1; i <= 40; i++)

            javaPrograming.enrolStudent();

        javaPrograming.displayInfor();

        javaPrograming.enrolStudent();

        javaPrograming.displayInfor();

    }

}
```

Truyền tham số cho phương thức

- Java sử dụng cách thức truyền theo tham trị (pass-by-value)
 - Khi truyền tham số có kiểu tham chiếu, một bản sao sẽ được tạo ra:
 - ✓ Giá trị tham chiếu gốc không bị thay đổi sau khi kết thúc phương thức
 - ✓ Giá trị của đối tượng giữ lại mọi thay đổi mà phương thức đã tạo ra
- hiệu quả tương tự như truyền tham biến

```
package java.oop.example;

/** The AnyValue class contains a integer */
public class AnyValue {
    private int data;

    /** Construcs a new object with initial value*/
    public AnyValue (int initData){
        this.data = initData;
    }

    /** Setter method*/
    public void setData (int newData){
        this.data = newData;
    }

    /** Getter method*/
    public int getData(){
        return this.data;
    }
}
```

Truyền tham số cho phương thức

```
package java.oop.example.passing;

import java.oop.example.AnyValue

/**This class illustrates passing parameter when calling a method */

public class PassingParameter {

    /** Method has a primitive value*/

    private static void changeNumber(int a){

        a = a + 1;

    }

    /** Method has a reference value*/

    private static void changeValue(AnyValue value){

        value.setData(10);

    }

}
```


Truyền tham số cho phương thức

```
public static void main(String[] args){
```

```
    //pass primitive parameter
```

```
    int number = 1;
```

```
    System.out.println("Before changing: number = "+ number);
```

```
    changeNumber(number);
```

```
    System.out.println("After changing: number = "+ number);
```

```
    //pass reference parameter
```

```
    AnyValue value = new AnyValue(1);
```

```
    System.out.println("Before changing: value has " + value.getData());
```

```
    changeValue(value);
```

```
    System.out.println("After changing: value has " + value.getData());
```

```
}
```

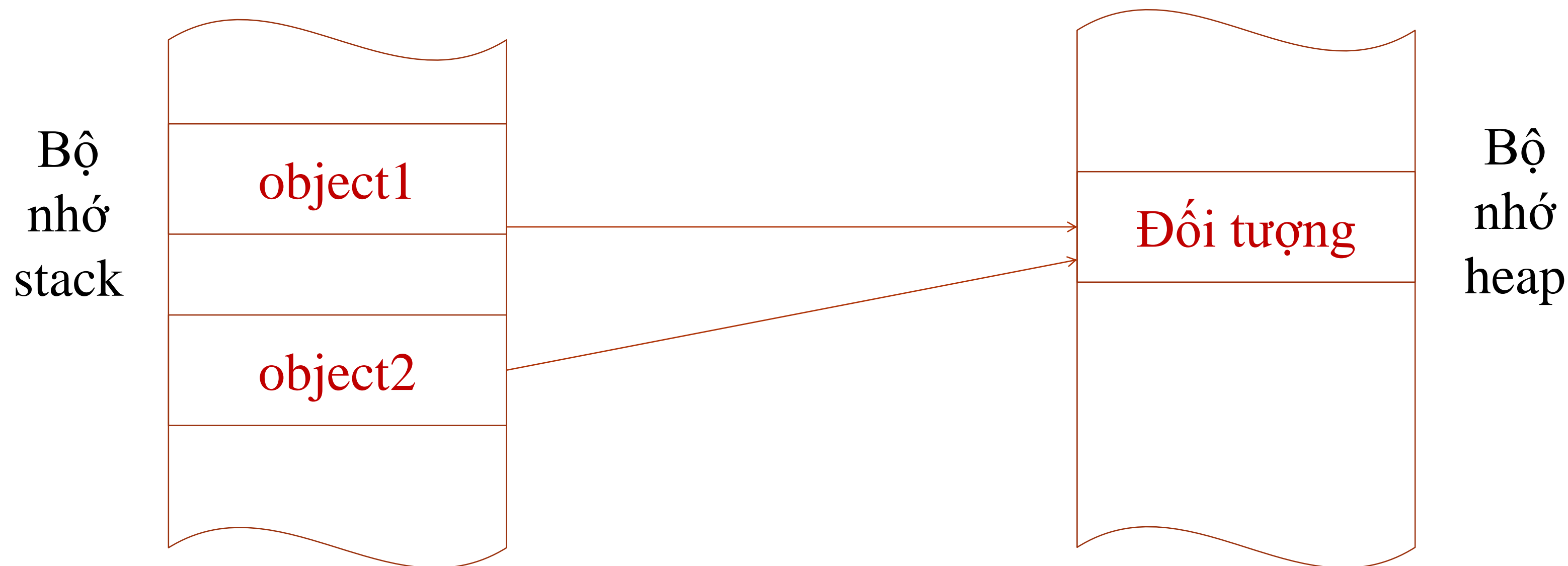
```
}
```

Phép gán đối tượng

- Java cho phép gán 2 đối tượng cho nhau:

object2 = object1

- Hãy nhớ rằng object1 và object2 chỉ là các tham chiếu. Do đó câu lệnh gán trên chỉ là gán giá trị tham chiếu. Hay nói cách khác, một thể hiện sẽ có hai tham chiếu object1 và object2



So sánh 2 đối tượng

- So sánh hai giá trị có kiểu nguyên thủy: ==
- So sánh hai đối tượng : **object1 == object2**?
 - ✓ Luôn nhớ rằng object1 và object2 chỉ là tham chiếu. Vì vậy toán tử == chỉ so sánh giá trị của 2 tham chiếu
 - ✓ Kiểm tra 2 tham chiếu có cùng trỏ đến một đối tượng không
- Phương thức equals(Object)
 - ✓ Các lớp của Java:
 - Trả về **true** nếu mọi thành viên của 2 đối tượng có giá trị bằng nhau
 - Trả về **false** nếu ngược lại
 - ✓ Các lớp định nghĩa mới: Cần sử dụng nguyên lý chồng phương thức để viết lại. Chúng ta sẽ đề cập đến sau.

Gán và so sánh 2 đối tượng

```
package java.oop.example.comparison;
import java.oop.example.AnyValue
/** The ObjectComparision class illutrates comparing two
object*/
public class ObjectComparison {
    public static void main(String[] args) {
        AnyValue value1 = new AnyValue(1);
        AnyValue value2 = new AnyValue(1);
        System.out.println("***Compare two object:");
        System.out.println("value1 == value2: " + (value1==value2));
        System.out.println("value1.equals(value2): " +
value1.equals(value2));
```

Gán và so sánh 2 đối tượng

```
System.out.println("***After assigning value2 to value1:");  
value1 = value2;  
System.out.println("value1 == value2: " + (value1==value2));  
value1.setData(100);  
System.out.println("The data of value1: " + value1.getData());  
System.out.println("The data of value2: " + value2.getData());  
}  
}
```

Mảng đối tượng

- **Cú pháp khai báo và khởi tạo:**

```
ClassName[] arrayName = new ClassName[size];
```

- **Trong đó:**

ClassName: Tên lớp mà đối tượng tạo ra từ lớp đó

arrayName: Tên mảng đối tượng

size: Kích thước

- **Các phần tử sẽ mang giá trị null**

Mảng đối tượng

- **Cú pháp khai báo và khởi tạo:**

```
ClassName[] arrayName = {new ClassName(),...,  
                           new ClassName()};
```

- Trong đó:

ClassName: Tên lớp mà đối tượng tạo ra từ lớp đó

arrayName: Tên mảng đối tượng

- Số lần gọi phương thức khởi tạo là kích thước của mảng



TÁI SỬ DỤNG MÃ NGUỒN

Tái sử dụng mã nguồn là gì?

- Sử dụng lại các mã nguồn đã viết
- Lập trình cấu trúc: chương trình con
- Lập trình hướng đối tượng: nhiều loại đối tượng có thuộc tính, hành vi tương tự nhau → tái sử dụng các lớp đã viết
 - ✓ Trong một lớp vẫn tái sử dụng phương thức
- **Ưu điểm:**
 - ✓ Giảm chi phí
 - ✓ Nâng cao khả năng bảo trì
 - ✓ Nâng cao khả năng mô hình hóa
 - ✓ ...

Các cách tái sử dụng mã nguồn

- Sao chép lớp cũ thành 1 lớp khác
 - Hạn chế: Dư thừa, khó quản lý khi có thay đổi
- **Kết tập:** Lớp mới là tập hợp hoặc sử dụng các lớp đã có
 - Chúng ta đã từng viết hàm main() trong đó có khai báo các đối tượng của một lớp. Nhưng đó không phải là kết tập
- **Kế thừa:** Lớp mới phát triển thêm các thuộc tính hoặc phương thức từ lớp đã có

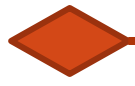


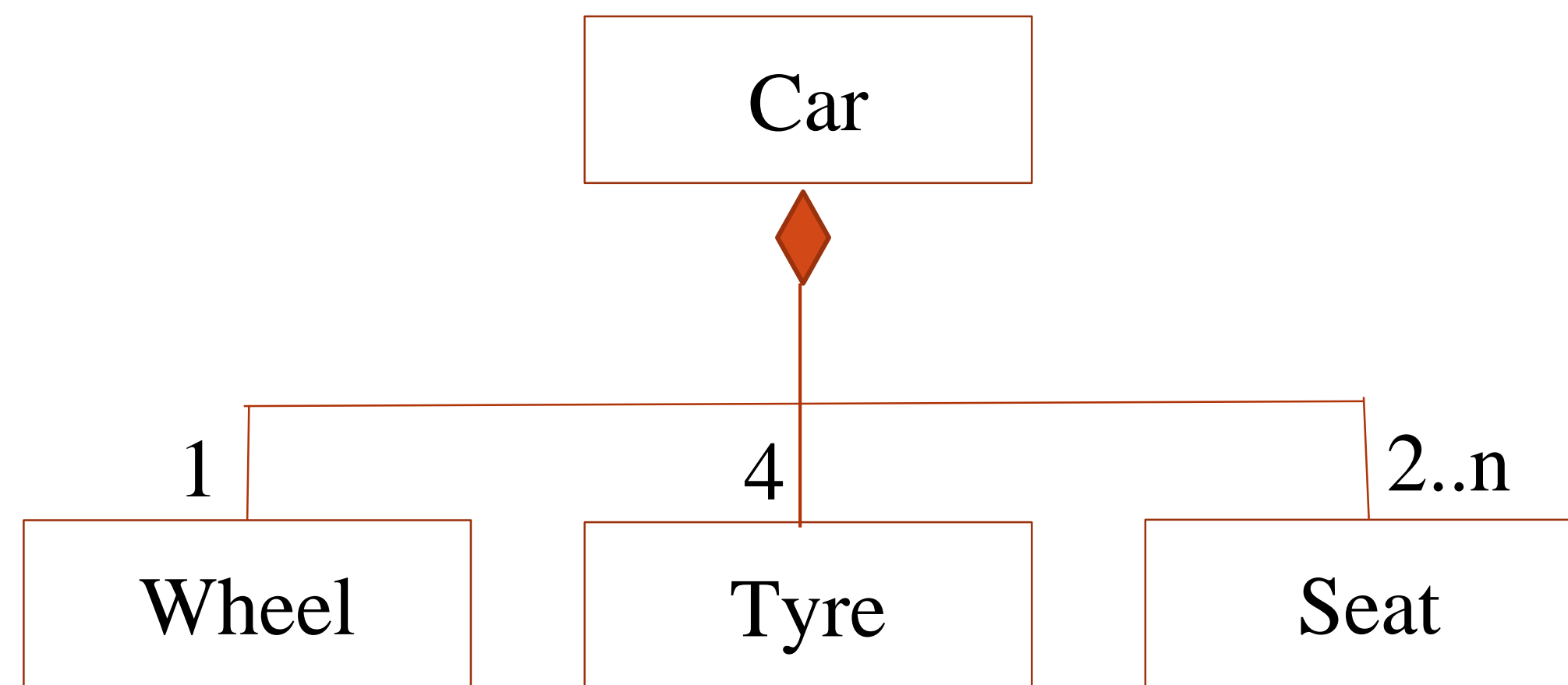
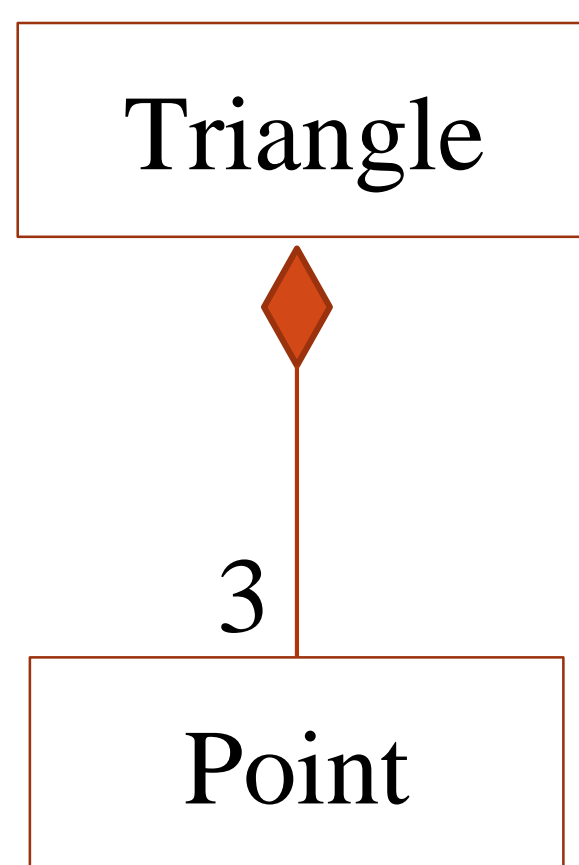
KẾT TẬP

Kết tập là gì?

- Thành phần lớp mới chứa các đối tượng của lớp cũ
- Lớp mới: Lớp chứa/Lớp toàn thể
- Lớp cũ: Lớp thành phần
- Ví dụ:
 - ✓ Lớp cũ: Điểm (Point)
 - ✓ Lớp mới: Tam giác (Triangle) có 3 điểm
- Lớp chứa tái sử dụng các **thuộc tính** và **phương thức** của lớp thành phần thông qua đối tượng

Biểu diễn kết tập trên biểu đồ thiết kế

- Lớp chứa  — Lớp thành phần
- Sử dụng bội số quan hệ:
 - ✓ 1 số nguyên dương (1, 2, 3...)
 - ✓ Dải số (0..1, 1..n)
 - ✓ Bất kỳ giá trị nào: *
 - ✓ Không ghi: mặc định là 1



Ví dụ - Lớp Point

```
package java.oop.basic.aggregation;

/** The Point class presents a point in the system Oxy */
public class Point {
    private double x, y;
    /** The constructor method
     * @param initX : the x coordinate
     * @param initY : the y coordinate
     */
    public void Point(double initX, double initY){
        this.x = initX;
        this.y = initY;
    }
}
```

Ví dụ - Lớp Point

```
/** The X setter method*/  
public void setX(double newX){  
    this.x = newX;  
}  
/** The Y setter method*/  
/** The X getter method*/  
public double getX(){  
    return this.x;  
}  
/** The Y getter method*/  
/** Display the coordinates of a point*/  
public void displayPoint(){  
    System.out.printf("(%f,%f\n",this.x, this.y);  
}
```

Ví dụ - Lớp Triangle

```
package java.oop.basic.aggregation;
/** The Triangle class presents a triangle in the system
Oxy */
public class Triangle {
    private Point vertex1, vertex2, vertex3;
    /** The constructor method
    * @param vertex1: the 1st coordinate
    * @param vertex2: the 2nd coordinate
    * @param vertex3 : the 3nd coordinate
    */
    public Triangle(Point vertex1, Point vertex2, Point vertex3){
        this.vertex1 = vertex1;
        this.vertex2 = vertex2;
        this.vertex3 = vertex3;
    }
}
```

Ví dụ - Lớp Triangle (tiếp)

```
/** The setter methods*/  
/** The getter method*/  
public void displayTriangle()  
    System.out.println("The triangle has three vertices:");  
    vertex1.displayPoint();  
    vertex2.displayPoint();  
    vertex3.displayPoint();  
}  
}
```

Ví dụ

- Xây dựng một trò chơi xúc xắc. Cách chơi như sau:
 - Mỗi hạt xúc xắc được gieo sẽ có giá trị ngẫu nhiên 1..6
 - Hai người lần lượt gieo 1 hạt xúc xắc
 - Sau mỗi lượt gieo, số điểm của lượt đó được tích lũy vào số điểm của người chơi
 - Sau các lượt gieo theo quy định, người thắng cuộc là người có tổng số điểm lớn hơn

Phát hiện lớp

- Trò chơi cần có 3 lớp: Die (xúc xắc), Player (người chơi), Match (trận đấu)
- Lớp Die:
 - Thuộc tính: face
 - Phương thức: roll() thiết lập một giá trị ngẫu nhiên cho face
- Lớp Player:
 - Thuộc tính: name, point
 - Phương thức: throwDie(Die) chờ nhấn phím bất kỳ để thực hiện gieo xúc xắc

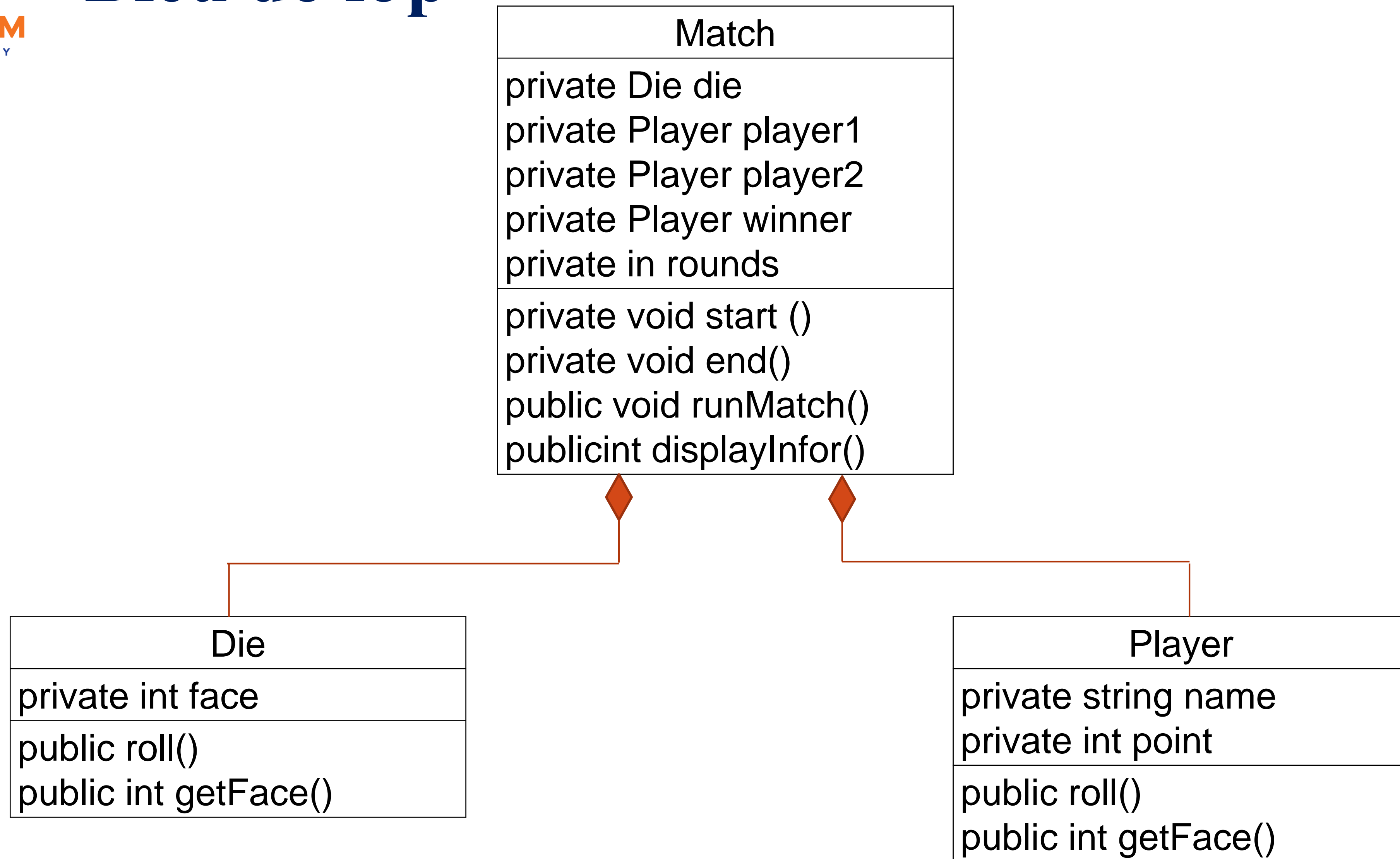
Phát hiện lớp (tiếp)

- Lớp Match:

- Thuộc tính: die, player1, player2, winner, rounds (số lượt gieo)
- Hành vi: start(), end(), runMatch(), displayInfor()

Match
private Die die private Player player1 private Player player2 private Player winner private in rounds
private void start () private void end() public void runMatch() publicint displayInfor()

Biểu đồ lớp



Lớp Die

```
package java.oop.die.game;
import java.util.Random;
/**
 * The Die class presents the die in game
 */
public class Die {
    private int face;
    /**
     * Constructs a new die
     */
    public Die(){
        this.face = 1;
    }
}
```

Lớp Die (tiếp)

```
/**
 * Generate randomly a face
 * @return The face of the dice after rolling
 */
public int roll(){
    Random rand = new Random();
    this.face = rand.nextInt(5) + 1;
    return this.face;
}

/**
 * Get the current face of the die
 * @return The current face
 */
public int getFace(){
    return this.face;
}
}
```

Lớp Player

```
package java.oop.die.game;
import java.io.IOException;
import java.util.Scanner;

/** This class describes a player in game */
public class Player {
    private String name;
    private int point;
    private Scanner pressKey;

    /**Constructs a new player with his name
     * @param initName: The player's name */
    public Player(String initName){
        this.name = new String(initName);
        this.point = 0;
    }
}
```


Lớp Player (tiếp)

```
/**Player throw a die
 * @param die: The Die object */
public void throwDie(Die die){
    int currentThrow;
    pressKey = new Scanner(System.in);
    System.out.print("Press Enter to throw your die!");
    try {
        System.in.read();
    } catch (IOException e) {
        e.printStackTrace();
    }
    pressKey.nextLine();
    currentThrow = die.roll();
    this.point += currentThrow;
    System.out.println(currentThrow + " points");
}
```

Lớp Player (tiếp)

```
/**Set a new point for player after he threw die
 * @param newPoint: The new point after throwing
 */
public void setPoint(int newPoint){
    this.point = newPoint;
}
/**Get the current point of the player
 * @return: The current point*/
public int getPoint(){
    return this.point;
}
/**Get the name of player
 * @return The name of player */
public String getName(){
    return this.name;
}
}
```

Lớp Match

```
package java.oop.die.game;
public class Match {
    private Die die;
    private Player player1, player2;
    private Player winner;
    private int rounds;
    /** Constructs a new match with initial values
     * @param initPlayer1: The 1st player
     * @param initPlayer2: The 2nd player
     * @param initRounds: The number of rounds
     */
    public Match(String initPlayer1, String initPlayer2, int
initRounds){
        this.player1 = new Player(initPlayer1);
        this.player2 = new Player(initPlayer2);
        this.die = new Die();
        this.rounds = initRounds;
    }
}
```

Lớp Match(tiếp)

```
/** Running the match */
public void runMatch(){
    this.start();
    this.stop();
    this.displayInfor();
}
/** Start the match*/
private void start(){
    System.out.println("Start!");
    for(int i = 1; i<= this.rounds; i++){
        System.out.println("-----Round " + i + "-----");
        System.out.println(player1.getName() + " throw!");
        player1.throwDie(die);
        System.out.println(player2.getName() + " throw!");
        player2.throwDie(die);
    }
}
```

Lớp Match(tiếp)

```
/** Stop the match */
private void stop(){
    int pointPlayer1, pointPlayer2;
    pointPlayer1 = player1.getPoint();
    pointPlayer2 = player2.getPoint();
    if(pointPlayer1 > pointPlayer2) this.winner = this.player1;
    else if(pointPlayer2 > pointPlayer1) this.winner = this.player2;
}

/** Display the information of the game */
public void displayInfor(){System.out.println("---RESULT---");
    System.out.println(player1.getName() + " has " +
        player1.getPoint() + " points");
    System.out.println(player2.getName() + " has " +
        player2.getPoint() + " points");
    if(this.winner!=null)
        System.out.println("The winner is " + winner.getName());
    else System.out.println("You are draw");
}
}
```

Lớp Game

```
package java.oop.die.game;
import java.util.Scanner;
public class Game {
    public static void main(String[] args) {
        Match match;
        String player1, player2;
        int rounds;
        Scanner inputData = new Scanner(System.in);
        System.out.print("Enter the name of the 1st player: ");
        player1 = inputData.next();
        System.out.print("Enter the name of the 2nd player: ");
        player2 = inputData.next();
        System.out.print("Enter the number of rounds: ");
        rounds = inputData.nextInt();
        match = new Match(player1, player2, rounds);
        match.runMatch();
    }
}
```