

پروانه سازی پیشرفته



چند ریختی

مهدی مصطفی زاده

سرفصل مطالب

- مفهوم چندریختی (polymorphism)
- کاربرد چندریختی
- مفاهیم همبستگی (cohesion) و جفت‌شدگی (coupling)
- اصول DIP و OCP
- رده‌ها و متدهای انتزاعی (abstract)
- متغیرها، رده‌ها و متدهای نهایی (final)
- مفهوم انقیاد پویا



مفهوم چندریختی

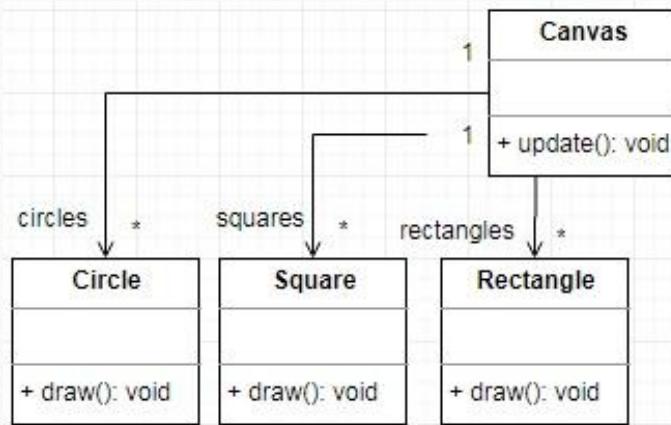


- رفتار «حرف زدن» در رده‌ی «حیوان» تعریف شده است.
- این رفتار، به همه‌ی زیررده‌ها (سگ، گربه، ...) به ارث می‌رسد.
- پس اگر مجموعه‌ای (آرایه‌ای) از حیوانات داشته باشیم، می‌توانیم آن را پیمایش کنیم و به یکایک حیوانات بگوییم: «حرف بزن!»
- خروجی: میو میو، واق واق، میو میو، قدقد، قوقولی قوقو ...
- خروجی کاملاً بستگی به نوع دقیق حیوانات دارد.



مثال

- برنامه‌ی گرافیکی (شبهه برنامه‌ی paint) را در نظر بگیرید.
- یک بوم نقاشی (Canvas) داریم.
- می‌توان به تعداد دلخواه دایره، مربع و مستطیل بر روی آن رسم کرد.



مثال (ادامه)

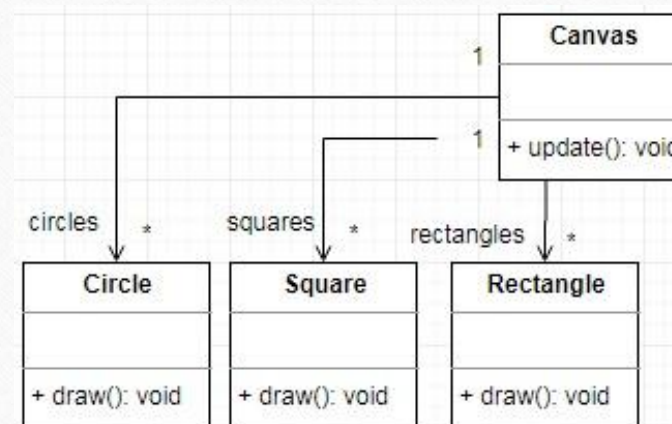
```
package graphics;

import java.util.ArrayList;

public class Canvas {
    ArrayList<Circle> circles = new ArrayList<>();
    ArrayList<Square> squares = new ArrayList<>();
    ArrayList<Rectangle> rectangles = new ArrayList<>();

    // other fields & methods

    public void update() {
        for(Circle circle : circles)
            circle.draw();
        for(Square square : squares)
            square.draw();
        for(Rectangle rectangle : rectangles)
            rectangle.draw();
    }
}
```



```
public class Circle {
    public void draw() {
        System.out.println("Circle");
    }
}
```



بدی راه حل ارائه شده

```
package graphics;

import java.util.ArrayList;

public class Canvas {
    ArrayList<Circle> circles = new ArrayList<>();
    ArrayList<Square> squares = new ArrayList<>();
    ArrayList<Rectangle> rectangles = new ArrayList<>();

    // other fields & methods

    public void update() {
        for(Circle circle : circles)
            circle.draw();
        for(Square square : squares)
            square.draw();
        for(Rectangle rectangle : rectangles)
            rectangle.draw();
    }
}
```

- فرض کنید بخواهیم شکل جدیدی (مثلاً) را به اشکال نرم افزار اضافه نماییم.

- بخش‌های مختلف رده‌ی بوم (Canvas) تغییر می‌یابد (متد update، فیلد جدید، متد جدید برای افزودن مثلاً).

- نادیده گرفتن OCP

Open to extension,
Closed to modification Principle



حل مثال با چندریختی

```
package graphics;

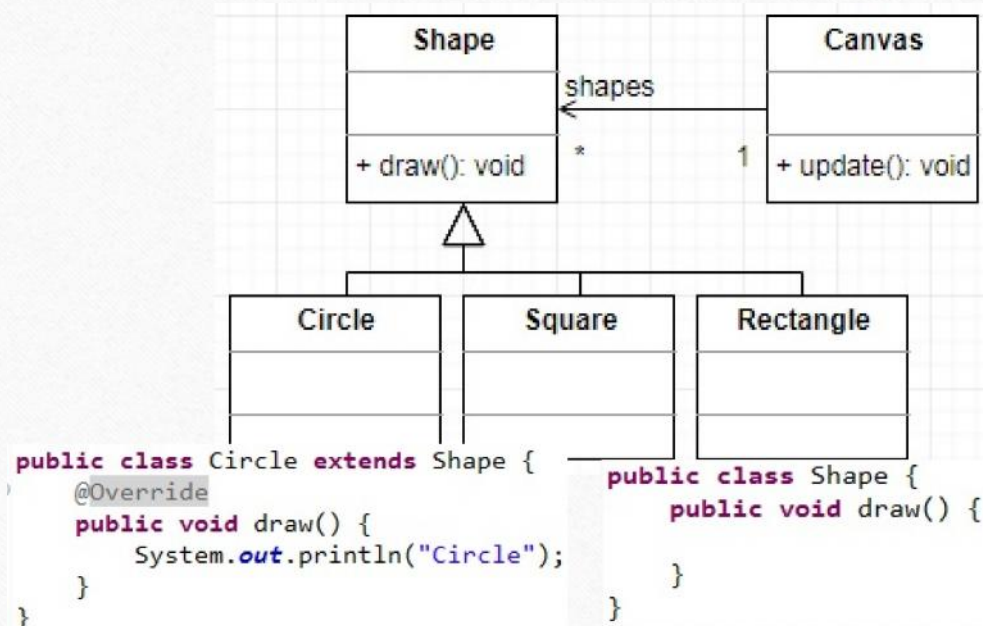
import java.util.ArrayList;

public class Canvas {
    ArrayList<Shape> shapes = new ArrayList<>();

    // other fields & methods

    public void addShape(Shape shape) {
        shapes.add(shape);
    }

    public void update() {
        for(Shape shape : shapes)
            shape.draw();
    }
}
```

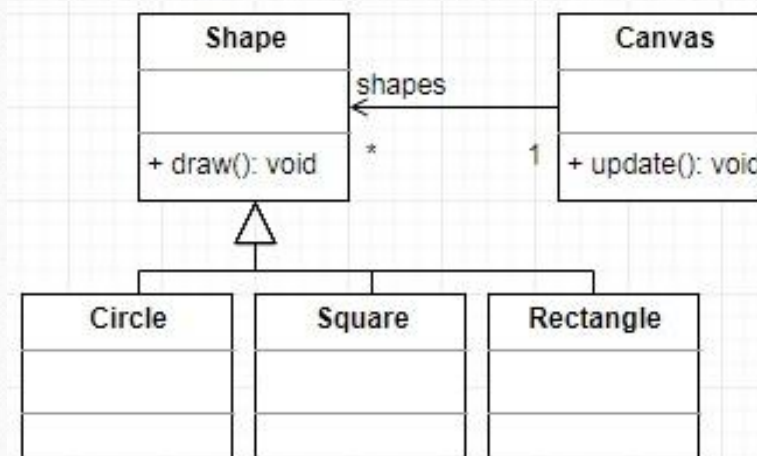


حل مثال با چندریختی (ادامه)

```
7 public static void main(String[] args) {  
8     Circle circle = new Circle();  
9     Shape shape = new Rectangle();  
10    Canvas canvas = new Canvas();  
11    canvas.addShape(circle);  
12    canvas.addShape(shape);  
13    canvas.update();  
14 }
```

Ma... Pr... Se... Da... Sn... Pr... Co... Pr...

<terminated> Hello [Java Application] C:\Program Files\Java\jdk1.8.0_112\bin\javaw
Circle
Rectangle



اصل وارونه‌سازی وابستگی‌ها

- در راه‌حل جدید (با استفاده از چندریختی)، توانستیم وابستگی‌ها را از رده‌های سطح پایین‌تر (concrete تر) به رده‌های سطح بالاتر (abstract تر) منتقل کنیم.
- DIP (Dependency Inversion Principle)
- DIP یکی از راه‌های برقراری OCP است.



تغییر نوع به بالا و به پایین

- تغییر نوع به بالا (UpCasting):

استفاده از شیء فرزند به عنوان شیء ای از ردهی پدر.

- تغییر نوع به پایین (DownCasting):

استفاده از شیء پدر به عنوان شیء ای از ردهی فرزند
(به شرطی که ممکن باشد).

```
8 public static void main(String[] args) {
9     Canvas canvas = new Canvas();
10    Circle circle = new Circle();
11
12    Shape shape = new Rectangle(); // valid (UpCasting)
13    canvas.addShape(circle); // valid (UpCasting)
14    shape = circle; // valid (UpCasting)
15
16    circle = (Circle) shape; // valid (DownCasting)
17    Square square = (Square) shape; // invalid (runtime error)
18
19    Student st = (Student) shape;
20
21    canvas.update();
22 }
```

Cannot cast from Shape to Student
Press 'F2' for focus



انقیاد پویا (Dynamic Binding)

```
Shape shape = new Rectangle();  
shape.draw();
```

```
Shape shape = new Square();  
shape.draw();
```

- قطعه کد مقابل را در نظر بگیرید. متد draw ردهی Shape فراخوانی می شود یا ردهی Rectangle؟

- یادآوری از اسلاید اول: خروجی کاملاً بستگی به نوع دقیق حیوانات دارد.

- نوع دقیق شیء رفتار را مشخص می کند، نه نوع ارجاع.

- چندریختی = واسط یکسان، اما رفتار متفاوت (دو قطعه کد را مقایسه نمایید).

- نوع دقیق شیء در زمان اجرا مشخص می شود (انقیاد پویا).



رده‌ها و متدهای انتزاعی (abstract)

- پیش از این، رده‌ی Shape را به شکل مقابل تعریف کرده بودیم. آیا پیاده‌سازی draw برای شیء موضوعیت دارد؟

```
public class Shape {  
    public void draw() {  
  
    }  
}
```

- پس آن را انتزاعی تعریف می‌کنیم. یعنی پیاده‌سازی آن را به تعریف رده‌های فرزند موکول می‌کنیم.

- اگر رده‌ای دارای حداقل یک متد انتزاعی باشد، الزاماً پایستگی خود رده را نیز انتزاعی کنیم.

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

- یک رده می‌تواند انتزاعی باشد؛ اما هیچ متد انتزاعی نداشته باشد. (به چه کار می‌آید؟)



رده‌ها و متدهای انتزاعی (ادامه)

- نمی‌توان از روی رده‌ی انتزاعی شی‌ای ایجاد کرد (چرا؟).
- رده‌ی فرزند یک رده‌ی انتزاعی:
پایستی همه‌ی متدهای انتزاعی رده‌ی پدر (اجداد) را پیاده‌سازی کند
یا
خودش نیز به شکل انتزاعی تعریف شود.
- سوال: چرا نیاز است واسطه‌ی draw را حتماً در رده‌ی Shape تعریف کنیم؟؟



متغیرها، رده‌ها و متدهای نهایی (final)

- متغیرهای نهایی (final)، متغیرهایی که مقدار آنها (پس از مقداردهی اولیه) قابل تغییر نیست (مقدارشان نهایی است).
 - فیلدها، پارامترها، متغیرهای محلی
- رده‌های نهایی (final)، رده‌هایی هستند که قابل گسترش (extend) نیستند (تعریفشان نهایی است).
- متدهای نهایی (final)، متدهایی هستند که در رده‌های فرزند قابل بازتعریف نیستند (تعریفشان نهایی است).




متغیرها، رده‌ها و متدهای نهایی (ادامه)


- متغیرها، رده‌ها و متدهای نهایی با کلیدواژه‌ی `final` تعریف می‌شوند.

- تفاوت متغیر نهایی (`final`) با متغیر رده‌ی غیرقابل تغییر (`immutable`) در چیست؟

```
final Person p = new Person("Ali")
p.setName("Ahmad");
p = new Person();
```

 The final local variable p cannot be assigned. It must be constant.

1 quick fix available:

 [Remove 'final' modifier of 'p'](#)

- `Final` بودن ناظر بر ارجاع است.

- `Immutable` بودن ناظر بر حالت شیء است.

- `Final` و `abstract` با هم سازگاری ندارند.



تعریف چند متد هم نام (Overload)

- می توان در یک رده، چند متد هم نام تعریف کرد؛ با این شرط که نوع و یا ترتیب پارامترهای آنها با هم متفاوت باشد (پارامترها و نه نوع خروجی).

```
public void addShape(Shape shape) {  
    System.out.println("shape");  
    shapes.add(shape);  
}
```

```
public void addShape(Polygon shape) {  
    System.out.println("polygon");  
    shapes.add(shape);  
}
```

```
public void addShape(Rectangle shape) {  
    System.out.println("rectangle");  
    shapes.add(shape);  
}
```

```
9      Canvas canvas = new Canvas();  
10     Shape shape = new Rectangle();  
11     canvas.addShape(shape);
```

<terminated> Hello [Java Application] C:\Program Files\Java\jdk1.8.0_111
shape



عملگر instanceof

- می‌توان با استفاده از آن بررسی کرد که آیا یک شیء از نوع یک رده است یا خیر.
- معمولاً قبل از تبدیل نوع (DownCasting)، از عملگر instanceof استفاده می‌شود.
- اگر شیء، نمونه‌ای از رده‌ی مورد نظر یا زیررده‌ای از آن باشد، true برمی‌گرداند.
 - اگر ارجاع به شیء null باشد، false برمی‌گرداند.
- اگر ارجاع به شیء و رده‌ی مورد نظر پدر و فرزند نباشند، خطای زمان ترجمه (کامپایل) رخ می‌دهد.



عملگر instanceof (مثال)

```
Shape shape = new Rectangle();
if(shape instanceof Circle) {
    Circle circle = (Circle) shape;
    // TODO
}
System.out.println(shape instanceof Rectangle); // true
System.out.println(shape instanceof Circle); // false
shape = null;
System.out.println(shape instanceof Rectangle); // false
System.out.println(shape instanceof Person);
```

✗ Incompatible conditional operand types Shape and Person

