

## دستور کار کارگاه برنامه‌نویسی پیشرفته

### جلسه پنجم

---

## آشنایی با مفاهیم وراثت و چندریختی

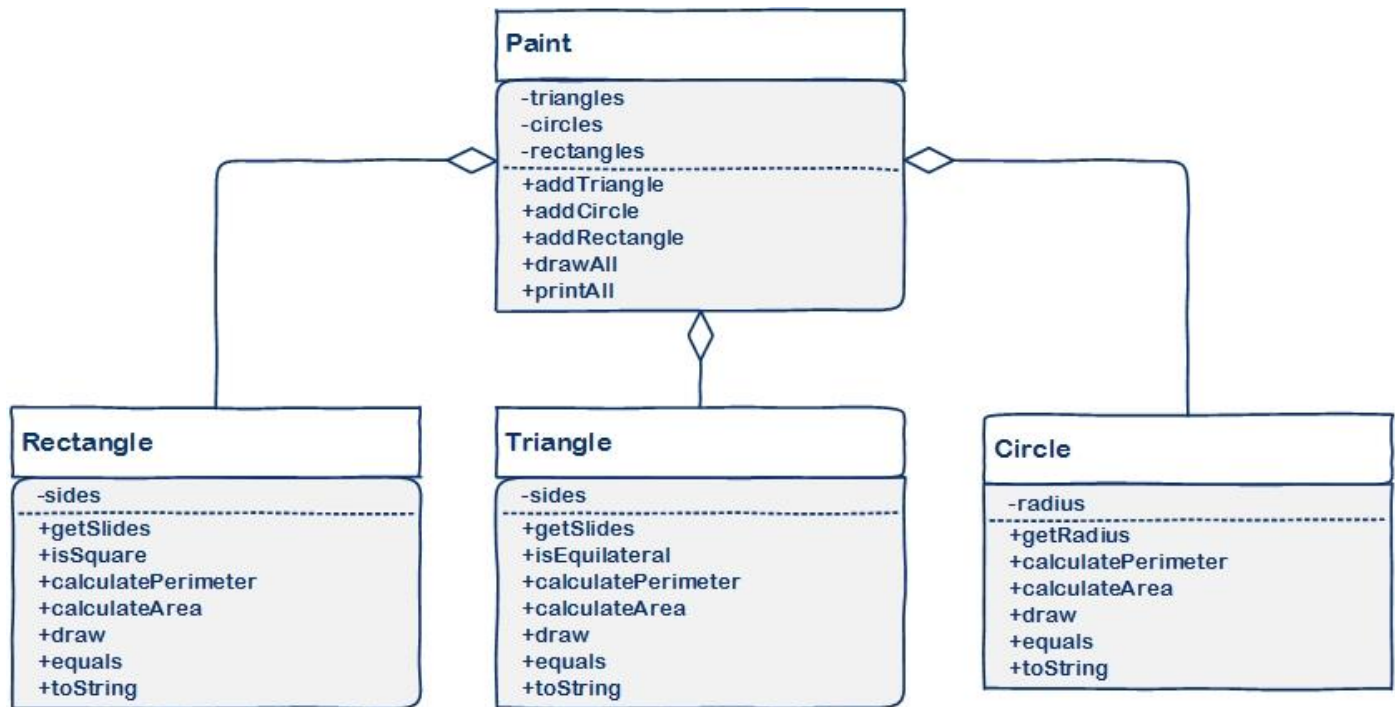
### مقدمه

در این جلسه می‌خواهیم با مفاهیم وراثت و چندریختی آشنا شویم. این دو مفهوم همانند بسیاری از مفاهیم برنامه‌نویسی شی‌گرا از دنیای واقعی الهام گرفته شده‌اند. به عنوان مثال، فرزندان یک خانواده را در نظر بگیرید. همه این فرزندان تعدادی ویژگی مشترک دارند که از والدین خود به ارث برده‌اند (مانند نفس کشیدن) و همچنین هر یک ویژگی‌های منحصر به فرد خود را دارند (مثلاً فرزندی توانایی خاصی در نواختن موسیقی داشته باشد در حالی که برادر/خواهرش فوتبالیست ماهر باشد). همین مفاهیم را می‌توان در برنامه‌نویسی شی‌گرایی نیز مدل و پیاده‌سازی کرد. یک کلاس می‌تواند از یک کلاس دیگر ویژگی‌هایی را به ارث ببرد و در عین حال ویژگی‌های منحصر به فرد خود را داشته باشد.

### مراحل انجام کار

حال برای تمرین مفاهیم وراثت و چندریختی قصد داریم تا یک برنامه نقاشی (Paint) طراحی و در آن شکل‌های مختلف هندسی را مدل‌سازی کنیم. برای این منظور ابتدا به کمک آن‌چه که از مفاهیم کلاس و شی در جلسات گذشته آموخته‌ایم، یک طراحی ساده از این مدل پیاده‌سازی خواهیم کرد.

در ابتدا پیاده‌سازی شما باید مطابق نمودار صفحه بعد باشد:



همان‌طور که در نمودار بالا مشاهده می‌کنید، هر کلاس شامل متغیرها و متدهای مخصوص به خود است (بالای خطچین متغیرها و پایین متدها؛ علامت - برای private و علامت + برای public). این ویژگی‌ها در بعضی کلاس‌ها مشترک و در بعضی منحصر به فرد هستند که در ادامه آن‌ها را بررسی می‌کنیم.

- متدهای مشترک میان سه کلاس **Circle**، **Triangle** و **Rectangle**:

۱. **calculatePerimeter**: متدی برای محاسبه محیط شکل (خروجی به صورت یک عدد double)

۲. **calculateArea**: متدی برای محاسبه مساحت شکل (خروجی به صورت یک عدد double)

۳. **draw**: متدی برای نمایش شکل (نوع شکل، محیط و مساحت آن را چاپ می‌کند)

۴. **equals**: متدی برای تشخیص یکسان بودن با شکلی دیگر (یک شکل هم‌نوع در ورودی گرفته و برابری آن را با خود بررسی می‌کند)

۵. **toString**: متدی برای توصیف شکل موردنظر به صورت یک رشته (نوع شکل و شعاع آن در دایره، نوع شکل و اضلاع آن در مثلث و مستطیل را به صورت یک رشته برمی‌گرداند)

## Circle

در کلاس دایره، متغیر radius برای ذخیره‌سازی شعاع دایره در نظر گرفته شده است که در constructor مقداردهی می‌شود. همچنین متدی برای دسترسی به مقدار آن باید ایجاد شود (getRadius).

## Triangle

در کلاس مثلث، متغیری از نوع ArrayList برای نگهداری اندازه اضلاع وجود دارد (sides) که در constructor مقداردهی می‌شود (constructor ۳ عدد را در ورودی می‌پذیرد و به sides اضافه می‌کند). همچنین این کلاس شامل متد getSides برای برگرداندن لیست اضلاع است. علاوه بر این، متدی به نام isEquilateral وجود دارد که متساوی‌الاضلاع بودن آن را بررسی و اعلام می‌نماید.

## Rectangle

در کلاس مستطیل، متغیری از نوع ArrayList برای نگهداری اندازه اضلاع وجود دارد (sides) که در constructor مقداردهی می‌شود (constructor ۴ عدد را در ورودی می‌پذیرد و در sides اضافه می‌کند). همچنین این کلاس شامل متد getSides برای برگرداندن لیست اضلاع است. علاوه بر این، متدی به نام isSquare وجود دارد که مربع بودن آن را بررسی می‌نماید.

## Paint

مطابق با نمودار، کلاس Paint سه ArrayList برای نگهداری اشیایی از جنس دایره، مثلث و مستطیل در خود دارد. در این کلاس سه متد برای اضافه کردن شیء به لیست شکل‌های متناظر وجود دارد. همچنین متد drawAll برای نمایش همه شکل‌های موجود در این کلاس در نظر گرفته است. در این متد با اجرای یک حلقه بر روی هر کدام از لیست‌ها، تمام شکل‌های موجود در آن لیست نمایش داده می‌شوند. متد printAll همانند drawAll با اجرای حلقه بر روی لیست‌های شکل‌ها، توصیف آن‌ها (نتیجه متد toString) را گرفته و چاپ می‌نماید.

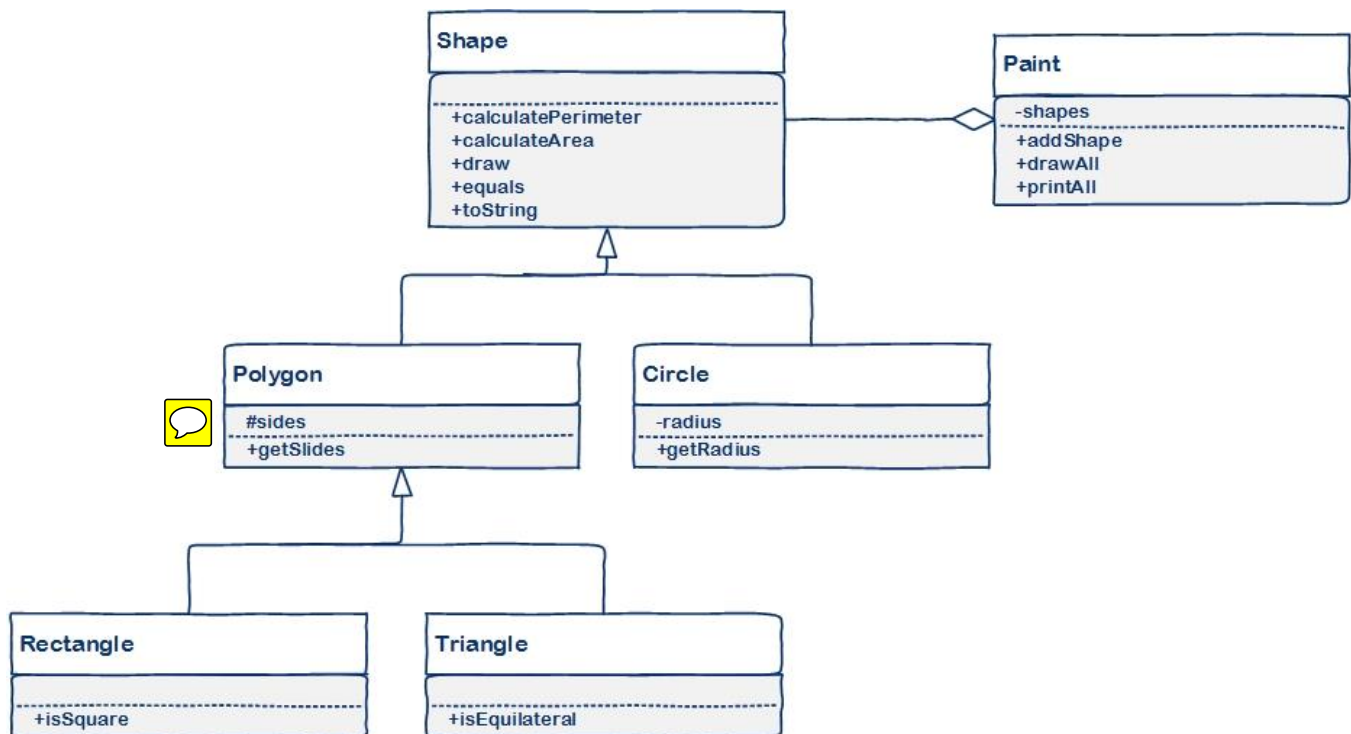
## انجام دهید

یک کلاس حاوی متد main بنویسید (مشابه کد زیر) و در آن یک شیء از کلاس Paint ایجاد نمایید. سپس با استفاده از متدهای اضافه کردن شکل‌ها، چند شکل از انواع دایره، مثلث و مربع به آن اضافه نمایید. در ادامه متدهای printAll و drawAll را از این شیء فراخوانی کنید و نتایج را به مدرس نشان دهید.

```
public static void main(String[] args) {  
    Circle circle1 = new Circle(19);  
    Circle circle2 = new Circle(3);  
  
    Rectangle rect1 = new Rectangle(1,4,1,4);  
    Rectangle rect2 = new Rectangle(8,5,8,5);  
    Rectangle rect3 = new Rectangle(6,6,6,6);  
  
    Triangle tri1 = new Triangle(2,2,2);  
    Triangle tri2 = new Triangle(4,4,6);  
  
    Paint paint = new Paint();  
  
    paint.addCircle(circle1);  
    paint.addCircle(circle2);  
    paint.addRectangle(rect1);  
    paint.addRectangle(rect2);  
    paint.addRectangle(rect3);  
    paint.addTriangle(tri1);  
    paint.addTriangle(tri2);  
    paint.drawAll();  
    paint.printAll();  
}
```

## اصلاح ساختار

در ادامه قصد داریم به کمک مفاهیم وراثت و چندریختی طراحی فوق را بهبود بخشیم. طراحی قبل یک مشکل اساسی دارد و علت آن وجود کدهای تکراری در کلاس‌های مختلف است. پیاده‌سازی این کدهای تکراری و تا حدود زیادی یکسان، وقت و انرژی شما را تا حد زیادی هدر می‌دهد. از طرف دیگر، در صورتی که بخواهیم شکل دیگری را نیز به مجموعه شکل‌ها اضافه کنیم، مجدداً باید بخشی از کدها را کپی کنیم. اما می‌توان با استفاده از وراثت و چندریختی تا حد مطلوبی از این موضوع جلوگیری کرد. وراثت، امکان استفاده مجدد از کدهای قبلی (code reusability) و وجود چند تابع با حالت‌های مختلف را فراهم می‌کند. بدین منظور در طراحی قبلی اندکی تغییرات ایجاد می‌کنیم و نمودار آن را به شکل زیر درمی‌آوریم.



همانطور که در نمودار بالا مشاهده می‌کنید، دو کلاس جدید به نام‌های **Shape** و **Polygon** (توضیحات آن در ادامه گفته شده است) اضافه می‌کنیم. همچنین متدهایی مثل `draw` و `calculateArea` از کلاس‌های شکل‌های هندسی حذف شده و تنها در کلاس **Shape** حضور دارند. علاوه بر این کلاس‌هایی مثل **Circle** و **Triangle** از کلاس والد (super class) با کلمه کلیدی **extends** ارث‌بری می‌کنند.

## Polygon

لیست اضلاع که در طراحی قبلی به صورت جداگانه در هر کدام از کلاس‌های Triangle و Rectangle قرار داشت، تنها در این کلاس قرار داده می‌شود و به این ترتیب از این کلاس ارث‌بری می‌کنند. همچنین لازم است متد toString که در کلاس Shape قرار گرفته است در این کلاس Override شود و رشته‌ای شامل تمام اضلاع چندضلعی برگردانده می‌شود (به عنوان مثال، برای مثلث با اضلاع ۸، ۱۰، ۴ باید رشته "side1:8, side2:10, side3:4" برگردانده شود). این کلاس در constructor خود یک آرایه از اعداد به صورت varargs دریافت کرده و پس از ایجاد لیست (ArrayList) sides، اعداد ورودی را در آن ذخیره می‌نماید. constructorهای کلاس‌های Triangle و Rectangle لازم است که اعداد دریافتی برای اضلاع را به کمک constructor کلاس Polygon در sides قرار دهند (یادآوری: AutoBoxing).

اما varargs چیست؟ این ویژگی در جاوا ۵ معرفی شده است و با ... (سه نقطه) مشخص می‌شود و به متد شما این امکان را می‌دهد که صفر یا بیشتر متغیر ورودی دریافت کند (تفاوت آن با آرایه چیست؟). به مثال زیر توجه کنید:

```
public void print(String... args) {
    for (String arg : args) {
        System.out.println(arg);
    }
}

print("hello", "folks");
```

AutoBoxing و Unboxing چیست؟ شما به طور ناخواسته از این مفهوم در این قسمت و پیاده‌سازی قبلی استفاده کرده‌اید. همانطور که می‌دانید تمام متغیرهای اولیه (primitive) دارای کلاس متناظر هستند (مثلا معادل int کلاس Integer). برای درک بیشتر، به مثال بعد توجه کنید.

به نظر شما چگونه یک شی Integer را توانستیم به ۲ تقسیم کنیم؟

```
public static int sumEven(List<Integer> li) {
    int sum = 0;
    for (Integer i : li) {
        if (i % 2 == 0)
            sum += i;
    }
    return sum;
}
```

## Shape

این کلاس متدهای مشترک موجود در کلاس‌های مربوط به شکل‌ها را در خود نگهداری می‌کند و دو کلاس Polygon و Circle از آن ارث‌بری می‌کنند. در ادامه کلاس‌های Circle، Triangle و Rectangle لازم است تا متدهای موجود در کلاس Shape را Override نمایند. علاوه بر این، متد toString در کلاس‌های Triangle و Rectangle لازم است تا رشته‌ی خروجی را به کمک فراخوانی همین متد از کلاس Polygon تولید نمایند (به عنوان مثال، با فراخوانی متد toString برای shape در صورتی که شکل مورد نظر مثلث باشد، باید رشته "Triangle:: side1:8, side2:10, side3:4" بازگردانده شود).

## Paint

در بازطراحی انجام شده، این کلاس به جای استفاده از سه لیست متفاوت برای نگهداری شکل‌ها، تنها یک لیست از Shape‌ها را در خود نگهداری می‌نماید و متدهای آن تنها با Shape سروکار خواهند داشت. (پس چگونه می‌توان از متدهای مخصوص یک شکل خاص مانند دایره استفاده کرد؟)

## انجام دهید

پیاده‌سازی جدید را انجام دهید و سپس کلاس حاوی متد main را که برای ساختار اولیه نوشته شد، به گونه‌ای تغییر دهید که همان عملیات را با ساختار جدید انجام دهد. همچنین یک متد جدید به نام describeEqualSides در کلاس Paint اضافه کنید که یک پیمایش روی تمام شکل‌ها (shapes)

انجام دهد و در صورتی که مربع یا مثلث متساوی‌الاضلاع بود (این موضوع را با کلمه کلیدی instanceof و توابع isSquare و isEquilateral تشخیص دهید) به کمک toString آن را چاپ کنید و نتایج را به مدرس گزارش کنید.

## انجام دهید

در پایین یک متد main داده شده است. از شما می‌خواهیم با توجه به مفاهیم وراثت و چندریختی (بخش دوم پیاده‌سازی) هر خط را توجیه و در صورت وجود خطا آن را تصحیح نمایید (این کار را بدون پیاده‌سازی انجام دهید).

```
public static void main (String... args) {
    Circle circle1 = new Shape(19);
    Shape circle2 = new Circle(3);
    Rectangle rect1 = new Triangle(1,4,1);
    Polygon rect2 = new Rectangle(8,5,8,5);
    Rectangle rec3 = new Shape(6,6,6,6);
    Polygon tri1 = new Triangle(2,2,2);
    Triangle tri2 = new Triangle(4,4,6);
    Shape tri3 = new Triangle(2,2,2);

    circle1 = circle2;
    rect2 = rect3;
    tri1 = tri3;
    cricle2 = tri3;
    tri3 = tri2;

    rect3 = new Shape(2,3,2);
    System.out.println(rect3.toString());
}
```



## معرفی ابزار اشکال‌یابی در IntelliJ

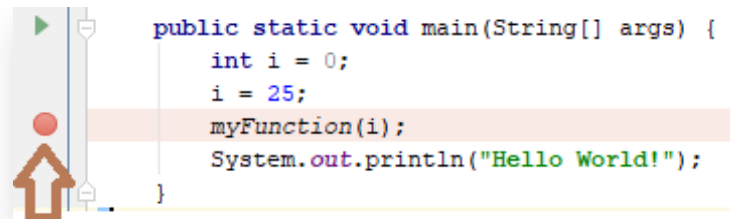
یکی از مهم‌ترین مهارت‌های برنامه‌نویسی، مهارت اشکال‌یابی و اشکال‌زدایی است که خود به عنوان یکی از مهارت‌های اصلی در زمینه تولید و توسعه نرم‌افزار محسوب می‌شود. در این بخش، قصد داریم امکانات اشکال‌یابی IntelliJ را مورد بررسی قرار دهیم. (برای مطالعه بیشتر: <https://www.jetbrains.com/help/idea/debug-tool-window.html>)

نرم‌افزار IntelliJ به برنامه‌نویسان این امکان را می‌دهد تا بتوانند برنامه خود را به صورت خط به خط اجرا کرده و مقدار تمامی متغیرهای موجود را در هر لحظه مشاهده کنند. برای استفاده از این امکان، باید یک نقطه وقفه (breakpoint) در برنامه قرار داده شود. پس از قراردادن این نقطه در برنامه، با اجرای برنامه در حالت اشکال‌یابی می‌توان از امکان اجرای خط به خط برنامه و مشاهده مقادیر متغیرها استفاده نمود.

برنامه زیر را در نظر بگیرید. در این برنامه، تابعی تعریف شده است که مقدار ورودی را یک واحد افزایش می‌دهد و سپس مقدار جدید را به همراه یک رشته کاراکتر چاپ می‌کند.

```
public class Main {  
  
    public static void myFunction(int i) {  
        i++;  
        System.out.println("The variable you passed was " + i);  
    }  
  
    public static void main(String[] args) {  
        int i = 0;  
        i = 25;  
        myFunction(i);  
        System.out.println("Hello World!");  
    }  
}
```

شما فرض کنید در خط ۱۱ یک نقطه وقفه قرار می‌دهید. برای قراردادن نقطه وقفه کافی است تا ناحیه مشخص شده در شکل بعد را فشار دهید تا یک علامت دایره قرمز نشان داده شود.



```
public static void main(String[] args) {  
    int i = 0;  
    i = 25;  
    myFunction(i);  
    System.out.println("Hello World!");  
}
```

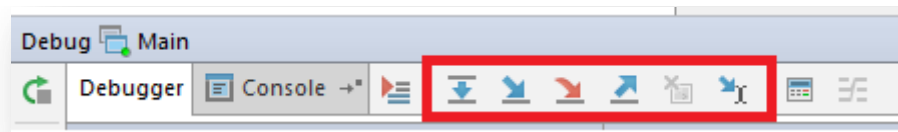
نحوه قراردادن نقطه وقفه

حال برای اجرای برنامه در حالت اشکالیابی کافی است مطابق با شکل زیر، در قسمت بالا و سمت راست صفحه کلیک کنید.



اجرای برنامه در حالت اشکالیابی

با کلیک کردن بر روی این دکمه، برنامه در حالت debugging اجرا می‌شود و بر روی خط دارای نقطه وقفه توقف می‌کند و صفحه Editor مقدار تمامی متغیرهای موجود در صفحه را نشان خواهد داد.



نوار ابزار اشکالیابی

در ادامه، در جدولی به معرفی هر یک از قسمت‌های نوار ابزار اشکالیابی می‌پردازیم.

کلید میان‌بر	کارکرد	دستور موجود در نوار ابزار
Alt + F10	خط فعلی اجرای دستورات را نشان دهد.	 <b>Show Execution Point</b>
F8	برنامه را تا خط بعدی اجرا کرده و توابع بینابینی را رد می‌کند.	 <b>Step Over</b>
F7	به داخل تابعی که در حال حاضر Debugger روی آن قرار دارد می‌رود.	 <b>Step Into</b>
Shift + Alt + F7	دستور Step Into بعضی اوقات دستورات مربوط به خود SDK را وارد نمی‌شود. با این کار شما Debugger را مجبور به ورود به این دستورات خواهید کرد.	 <b>Force Step Into</b>
Shift + F8	با اجرای این دستور Debugger از تابع فعلی خارج خواهد شد و به خط بعدی نقطه فراخوانی آن خواهد رفت.	 <b>Step Out</b>
Alt + F9	تا اجرای دستور در نقطه‌ی مکان‌نما در Editor پیش خواهد رفت.	 <b>Run to Cursor</b>

## انجام دهید

اکنون با توضیحات داده‌شده، طوری دستورات نوار ابزار را اجرا کنید که به داخل تابع رفته و تنها دستور اول آن را اجرا کند و سپس به دستور بعد از نقطه وقفه برود.

**سوال:** آیا می‌دانید چرا به اشکال‌های نرم‌افزاری اصطلاحاً bug گفته می‌شود؟