



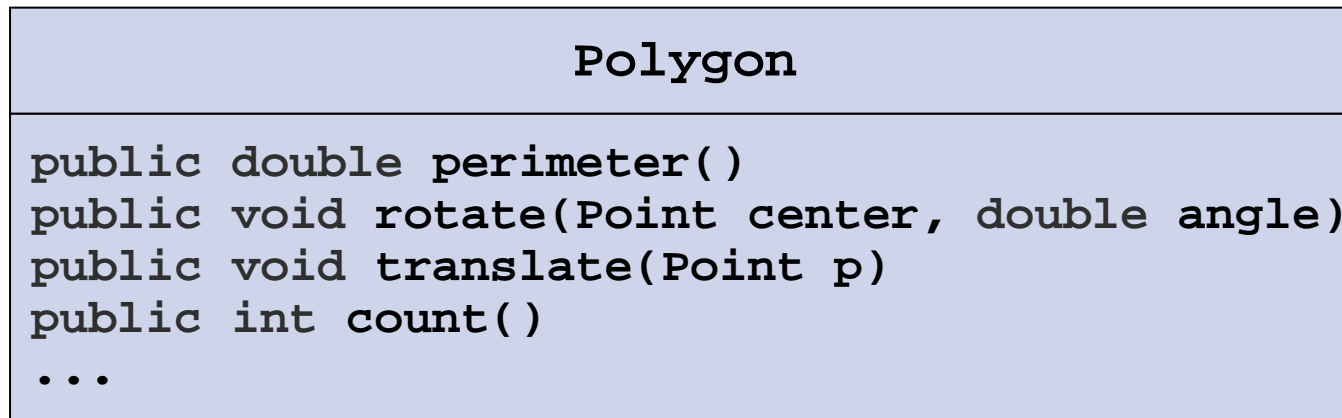
Software 1 with Java

Recitation No. 8
Inheritance (cont.), Exceptions

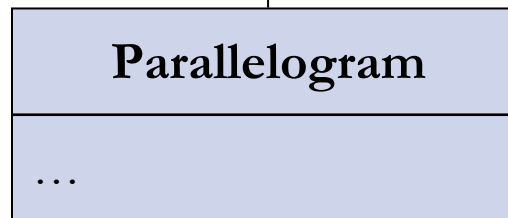
Class Parallelogram

■ מקבילית היא סוג של פוליגון

■ Parallelogram היא מקרה פרטי של Polygon



קשר ירשה
ב-JAVA הרחבה (extension)

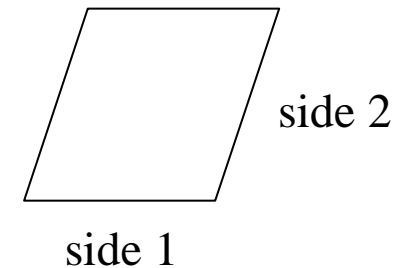


כל תכונות המצולע (המתודות)
תקפות למקבילית

Class Parallelogram

```
/**
 * @inv count() == 4
 * @inv vertices.get(0).distance(vertices.get(1)) ==
 *     side1
 * Similar invariants for the other 3 sides.
 */
public class Parallelogram extends Polygon {
    ...

    /** The two side lengths */
    private double side1, side2;
}
```



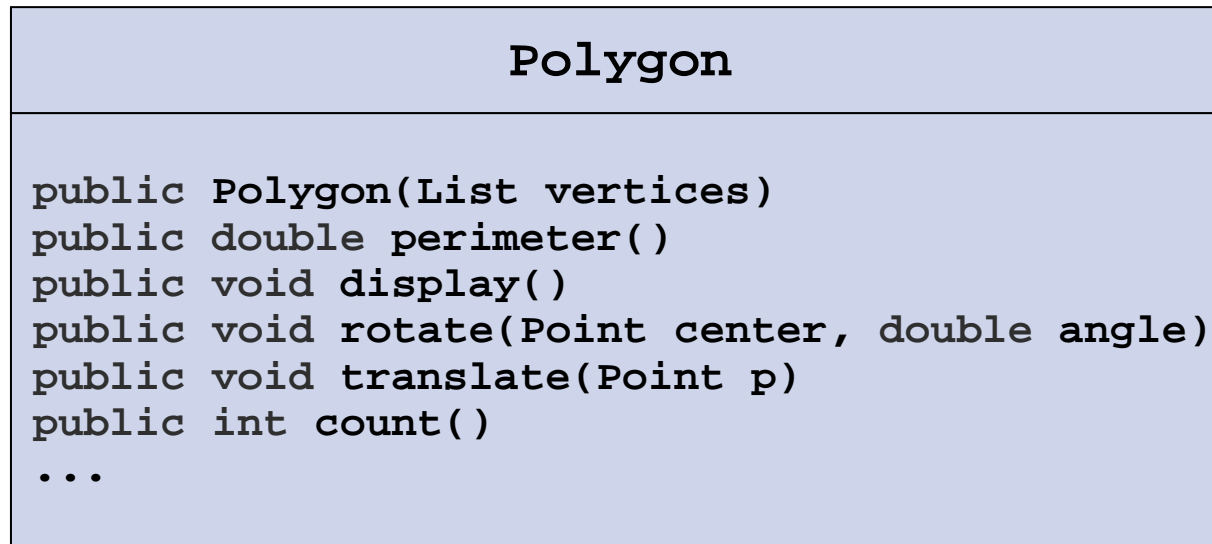
The Constructor

```
public class Parallelogram extends Polygon {
    /**
     * @pre vertices.size() == 4
     * @pre vertices.get(0).distance(vertices.get(1))
     * == vertices.get(2).distance(vertices.get(3))
     * @pre vertices.get(1).distance(vertices.get(2))
     * == vertices.get(0).distance(vertices.get(3))
     */
    public Parallelogram(List vertices) {
        super(vertices);
        // Setting side1, side2
        ...
    }
    ...
}
```

Overriding `perimeter()`

```
public class Parallelogram extends Polygon {  
    ...  
  
    /** Returns the polygon's perimeter */  
    public double perimeter() {  
        return 2 * (side1 + side2);  
    }  
  
    ...  
}
```

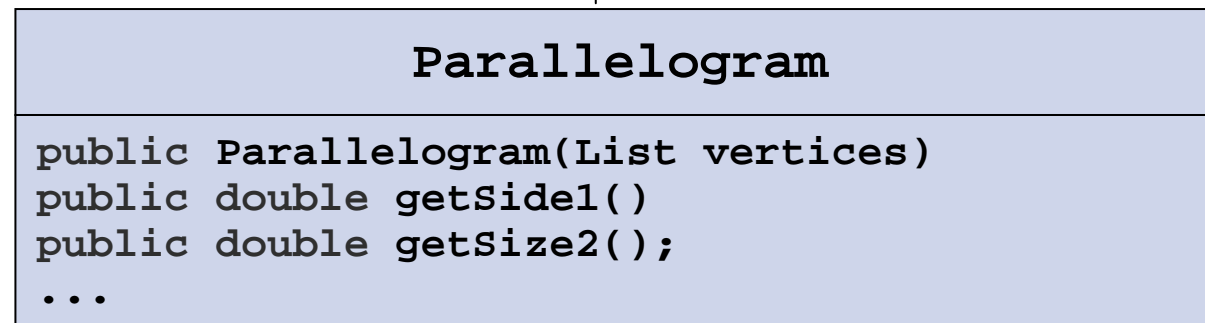
Inheritance Terms



הורה
מחלקת בסיס (base)
מחלקת על (super class)



קשר ירשה
ב-JAVA הרחבה (extension)



צאצא
מחלקה נגזרת (derived)
תת מחלקה (subclass)

private vs. protected

- השדה `vertices` הוגדר כ-`private`. הדבר מונע מ-`Parallelogram` גישה ישירה לשדה זה
- ניתן היה להגדיר שדה זה כ-`protected` וכך לאפשר ל-`Parallelogram` גישה ישירה
- שתי הגישות מקובלות ולשתיהן נימוקים טובים
- הבחירה בין שתי הגישות היא פרגמטית ותלויה בסיטואציה

private vs. protected

protected ■

■ `Parallelogram is a Polygon` - הוא עומד
ב"מבחן ההחלפה" ולכן צריך להיות עם אותן הזכויות.

private: ■

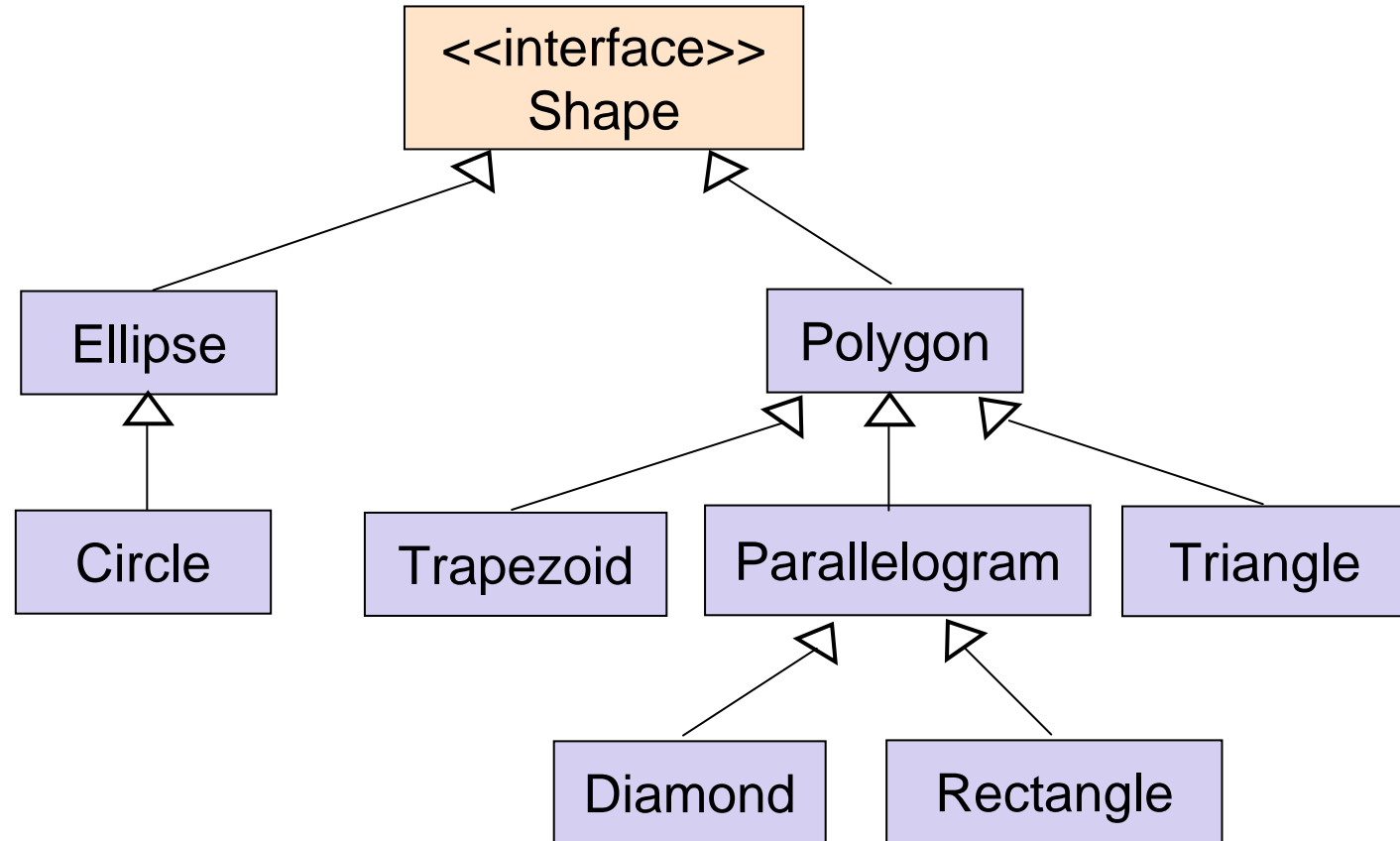
■ כשם שאנו מסתירים מלקוחותינו את המימוש כדי להגן
על שלמות המידע עלינו להסתיר זאת גם מצאצאנו
(איננו מכירים את יורשנו כפי שאיננו מכירים את
לקוחותינו)

■ צאצא עם עודף כח עלול להפר את חוזה מחלקת
הבסיס, להעביר את עצמו ללקוח המצפה לקבל את
אביו ולשבור את התוכנה

היררכית מחלקות ומנשקים

- איך נתמוך בצורות הנדסיות מישוריות נוספות כמו מלבן, מעגל, אליפסה, משולש, טרפז?
- לכל הצורות יש מנשק בסיסי משותף (היקף, שטח, סיבוב, הזזה וכד')
- ישנן תכונות שמשותפות רק לחלק מהצורות
דוגמא: זוויות ישרות לריבוע למלבן
- נגדיר מחלקות חדשות עם קשרי ירושה

היררכית מחלקות ומנשקים



רב צורתיות (Polymorphism)

- היכולת של הפנייה (reference) להתייחס בזמן ריצה לעצמים ממחלקות שונות
- תכונה זו מושגת בעזרת ירושה
- העצם המוצבע אינו משנה את טיפוסו. הפנייה מטיפוס מסוים עשויה להצביע בפועל לטיפוס ממחלקה נגזרת

Polymorphism Example

```
void foo(Polygon p, Rectangle r, Triangle t) {
    Polygon    localP;
    Rectangle  localR;
    Triangle   localT;

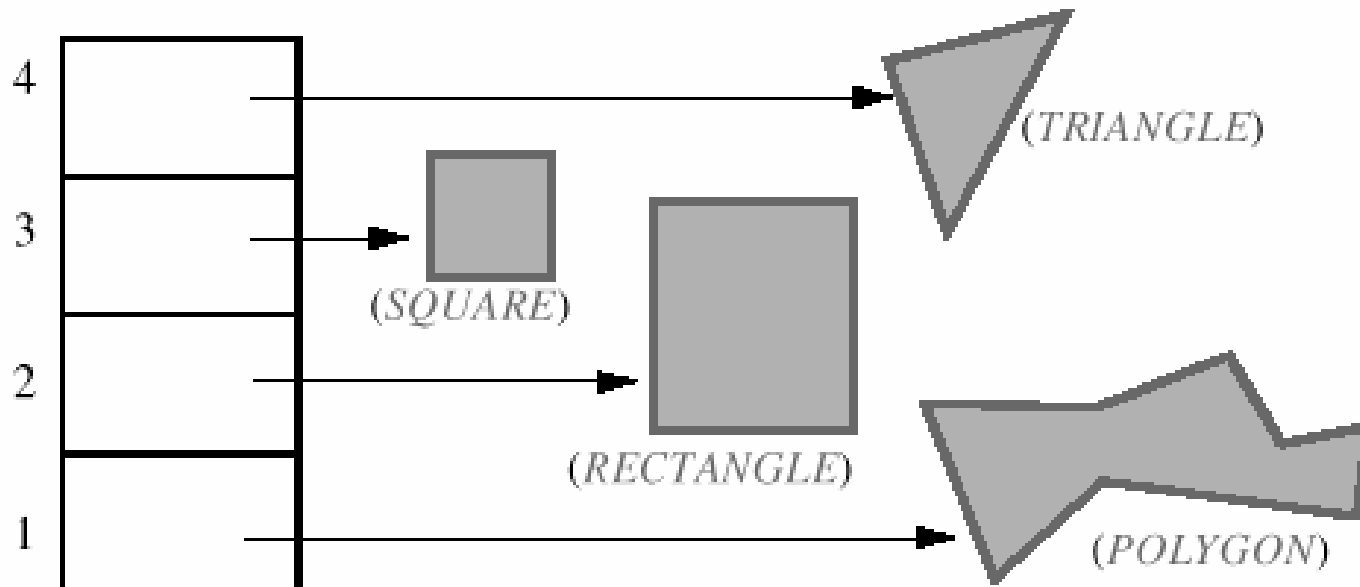
    localP = p;
    localP = r;
    localP = t;

    localR = p; // ERROR
    localR = r;
    localR = t; // ERROR

    localT = p; // ERROR
    localT = r; // ERROR
    localT = t;
}
```

מבנה נתונים פולימורפי

- מבנה נתונים המכיל אברים מטיפוסים שונים, אשר כולם צאצאים של אב משותף
- לדוגמא: מערך של מצולעים המכיל בפועל סוגים של מצולעים



מבנה נתונים פולימורפי

```
void foo(Polygon p, Rectangle r, Triangle t, Square s) {  
    Polygon[] polygons = new Polygon[4];  
  
    polygons[0] = p;  
    polygons[1] = r;  
    polygons[2] = s;  
    polygons[3] = t;  
  
    for(Polygon polygon : polygons) {  
        System.out.println(polygon.perimeter());  
    }  
}
```

עבור כל אחד מאברי המערך תופעל המתודה
perimeter ה"מתאימה" לפי טיפוס העצם המוצבע

העברת ארגומנטים למתודה

- השמה של הפניות מתרחשת בצורה מרומזת בכל פעם שאנו מעבירים הפנייה כארגומנט לפונקציה
- זוהי השמה של הטיפוס האקטואלי לטיפוס הפרמטר הפורמאלי
- גם השמה זו צריכה לציית לכללי הפולימורפיזם

```
void foo(Polygon p) { ... }
```

```
void bar() {  
    Rectangle r = new Rectangle(...);  
    foo(r);  
}
```

העברת ארגומנטים למתודה

```
void expectPolygon(Polygon p);  
void expectRectangle(Rectangle r);
```

```
void bar() {  
    Polygon p;  
    Rectangle r;  
    Triangle t;  
  
    expectPolygon(p);           // OK  
    expectPolygon(r);         // Also good  
    expectRectangle(r);       // OK  
    expectRectangle(p);       // Error  
    expectRectangle(t);       // Error  
}
```


זימון מתודה - דוגמא

```
void foo(Polygon polygon,  
         Parallelogram parallelogram) {  
    ✓ polygon.perimeter();  
    ✓ parallelogram.perimeter();  
    ✗ polygon.getSide1();  
    ✓ parallelogram.getSide1();  
}
```

טיפוס סטטי ודינמי

- **טיפוס של עצם:** טיפוס הבנאי שלפיו נוצר העצם. טיפוס זה קבוע ואינו משתנה לאורך חיי העצם.
- לגבי הפניות (references) לעצמים מבחינים בין:
 - **טיפוס סטטי:** הטיפוס שהוגדר בהכרזה על ההפניה
 - **הטיפוס הדינאמי:** טיפוס העצם המוצבע
 - הטיפוס הדינאמי חייב להיות נגזרת של הטיפוס הסטטי

```
Polygon p = new Rectangle(...);
```

טיפוס הסטטי של ההפניה

טיפוס העצם
הטיפוס הדינמי של ההפניה

טיפוס סטטי ודינמי של הפניות

```
void expectPolygon(Polygon p);  
void expectRectangle(Rectangle r);
```

```
void bar() {  
    Polygon p = new Polygon(...);  
    Rectangle r = new Rectangle(...);
```

```
     p = r;  
     expectRectangle(r);  
     expectPolygon(r);  
     expectPolygon(p);  
     expectRectangle(p);  
}
```

The static type of p remains Polygon.
Its dynamic type is now Rectangle.

Compilation Error despite that the
dynamic type of p is Rectangle

Exceptions

```
public static void main(String[] args) {  
    int i=1, j=1;  
    try {  
        i++;  
        j--;  
        if (i/j > 1)  
            i++;  
    } catch(ArithmeticException e) {  
        System.out.println(1);  
    } catch(Exception e) {  
        System.out.println(2);  
    } finally {  
        System.out.println(3);  
    }  
}
```

The output is:

1
3

Exceptions

```
public static void main(String[] args) {  
    int i=1, j=1;  
    try {  
        i++;  
        j--;  
        if (i/j > 1)  
            i++;  
    } catch(Exception e) {  
        System.out.println(1);  
    } catch(ArithmeticException e) {  
        System.out.println(2);  
    } finally {  
        System.out.println(3);  
    }  
}
```

Compilation Error:

Unreachable catch block. It is already handled by the catch block for Exception

Exceptions

```
public static void main(String[] args) throws Exception {
    int i=1, j=1;
    try {
        i++;
        j--;
        if (i/j > 1)
            i++;
    } catch(ArithmeticException e) {
        System.out.println(1);
        throw e;
    } catch(Exception e) {
        System.out.println(2);
    } finally {
        System.out.println(3);
    }
}
```

```
1
3
Exception in thread "main"
java.lang.ArithmeticException: / by zero
```

Method Overloading & Overriding

```
public class A {  
    public float foo(float a, float b) throws IOException {  
    }  
}
```

```
public class B extends A {  
    ...  
}
```

Which of the following methods can be defined in B:

1. float foo(float a, float b) {...}
2. public int foo(int a, int b) throws Exception {...}
3. public float foo(float a, float b) throws Exception {...}
4. public float foo(float p, float q) {...}

Answer: 2 and 4

Method Overriding

```
public class A {  
    public void print() {  
        System.out.println("A");  
    }  
}  
  
public class B extends A {  
    public void print() {  
        System.out.println("B");  
    }  
}
```

```
public class C {  
    public static void main(String args[]) {  
        B b = new B();  
        A a = b;  
  
        b.print();  
        a.print();  
    }  
}
```

Casting is
unnneeded

The output is: B
B
compile? If no, why?
throw a runtime exception?
no, what is the output?