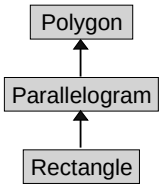
מחלקות על (super classes) ותתי מחלקות (sub classes)



- Polygon ו- Parallelogram הן **מחלקות על** של Rectangle
- ב- Java המחלקה Object היא **מחלקת על** של כל המחלקות (מלבד עצמה)
- Polygon היא **מחלקת על ישירה** של Parallelogram, Parallelogram היא **מחלקת על ישירה** של Rectangle
- Parallelogram ו-Rectangle הן **תתי מחלקות** של Polygon.

ירושה וחוזים

- תנאי קדם, תנאי בתר ושמורות שהוגדרו עבור מחלקה או מנשק תקפים גם לגבי צאצאי המחלקה (ומממשי המנשק)
- תת מחלקה יכולה:
  - להחליש תנאי קדם (לדרוש פחות) – אסור להוריד הרשאות (מותר להעלות)
  - לחזק את השמורה (להוסיף אילוצים)
  - **לחזק תנאי בתר** (להבטיח יותר): ערך החזר ספציפי יותר (String במקום Object)
  - **חיזוק תנאי צד**: הכרזת throws ספציפית יותר (IOException במקום Exception), או ביטול הכרזה

ירושה ב- Java

ירושה מתודות ושדות

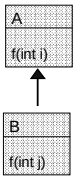
- מחלקה יורשת ממחלקת העל הישירה שלה וממנשקי העל הישירים שלה את כל המשתנים והמתודות שהם לא פרטיים (private) ונגישים לה, תחת האילוצים הבאים:
  - מתודות מופע – בתנאי שהן לא **נדרסו** (overridden) בה.
  - משתנים – בתנאי שהם לא **מוסתרים** (hidden) בה.
  - מתודות מחלקה – בתנאי שהן לא **מוסתרות** (hidden) בה.

בנאים

- האם יש מחלקות ללא בנאים?
  - לא. אם במחלקה A לא מצוין בנאי, יוסף בנאי ברירת המחדל: public A(){super();}
- מתי אין קריאה מבנאי לבנאי של מחלקת העל הישירה?
  - רק במחלקה Object . אחרת כל בנאי קורא לבנאי של מחלקת העל הישירה שלו לפני תחילת ביצועו (הפקודה הראשונה בבנאי). אם בקוד לא צוינה קריאה – הקומפיילר יוסיף קריאה ל- super().
- מה אם הבנאי הנקרא של מחלקת העל לא קיים או אין גישה אליו?
  - שגיאת קומפילציה.

דריסה (override) ע"י מתודות מופע

- חתימה של מתודה: השם+רשימת הטיפוסים של הפרמטרים (ערך ההחזר אינו חלק מהחתימה)
- מתודת **מופע** m2 במחלקה B דורסת (overrides) מתודה m1 במחלקה A (A≠B) אם:
  - B היא תת מחלקה של A
  - ל- m1 ו- m2 יש **חתימה זהה**
  - מתקיים לפחות אחד מהבאים:
    - m1 אינה פרטית ונגישה למחלקה של B.
    - m2 דורסת את m3 במחלקה C (C ≠A,B) ו- m3 דורסת את m1 (הגדרה רקורסיבית)



דריסה (override) ע"י מתודות מופע

- אם m2 דורסת את m1 היא חייבת לקיים את החוזה של m1:
  - הרשאה גבוהה או שווה
  - טיפוס החזר זהה או ספציפי יותר:
    - String במקום Object
    - SortedMap<Object> במקום Map<Object>
  - הכרזת throws זהה או ספציפית יותר / ביטול הכרזה

מתודות מופע (לא סטטיות)

```
class A1{
   protected A1 f(){System.out.println("A1.f()"); return g();}
   protected A1 g(){System.out.println("A1.g()"); return this;}
}

public class B1 extends A1{
   public A1 f(){System.out.println("B1.f()"); return super.f();}
   public B1 g(){System.out.println("B1.g()"); return this;}

   public static void main(String[] args){
        A1 a = new B1();
        a.f();
   }
}
```

מהו הטיפוס הסטטי והדינמי של this במהלך הפעלת המתודות?

B1.f()
B1.g()

פולימורפיזם

מתודות מופע (לא סטטיות)

```
class A1{
   protected A1 f(){System.out.println("A1.f()"); return g();}
   private   A1 g(){System.out.println("A1.g()"); return this;}
}

public class B1 extends A1{
   public A1 f(){System.out.println("B1.f()"); return super.f();}
   public B1 g(){System.out.println("B1.g()"); return this;}

   public static void main(String[] args){
        A1 a = new B1();
        a.f();
   }
}
```

מהו הטיפוס הסטטי והדינמי של this במהלך הפעלת המתודות?

B1.f()
A1.f()
A1.g()

הסתרה ופולימורפיזם

- האם יש פולימורפיזם בהפעלת מתודות סטטיות?
  - לא. הפעלת המתודה היא לפי הטיפוס הסטטי. בכדי למנוע בלבול בקריאת הקוד רצוי לציין את שם המחלקה בעת הקריאה למתודה.
- האם יש פולימורפיזם בגישה למשתנים?
  - לא. הגישה למשתנה היא ע"פ הטיפוס הסטטי.
  - ככלל, רצוי לא לתת שמות זהים למשתנים במחלקות בעלות קשר ירושה בכדי לא ליצור בלבול.

הסתרה ודריסה

- האם מתודת מחלקה יכולה להסתיר מתודת מופע?
  - לא. שגיאת קומפילציה.
- האם מתודת מופע יכולה לדרוס מתודת מחלקה?
  - לא. שגיאת קומפילציה.
- האם משתנה מופע יכול להסתיר משתנה מחלקה?
  - כן.
- האם משתנה מחלקה יכול להסתיר משתנה מופע?
  - כן.

Final

- Final Classes – לא ניתן להרחיב (שגיאת קומפילציה).
- Final Method – לא ניתן לבצע override (ישים עבור מתודות מופע בלבד) – מאפשר לקומפיילר לבצע אופטימיזציה לקריאה (יעילות)
- מחלקה/מתודה מופשטת (abstract) לא יכולה להיות final⇐.
- Final variable – השמה יחידה. עבור משתני מופע/מחלקה – חובה לאתחל (אין אתחול אוטומטי)

משתנה Final

- אם משתנה ה- final הוא reference לאובייקט. האם מותר לשנות את שדות האובייקט?
  - כן.
- אם משתנה ה- final הוא שדה מופע/מחלקה:
  - האם חייבים לאתחל אותו או שיש אתחול אוטומטי?
  - אין אתחול אוטומטי. חייבים לאתחל.
  - האם חייבים לאתחל בעת ההגדרה או שניתן לדחות את האתחול?
  - לא חייבים לאתחל בעת ההגדרה (blank final). במקרה זה: משתנה מחלקה צריך להיות מאותחל בבלוק אתחול סטטי {}(static); משתנה מופע צריך להיות מאותחל בסיום כל אחד מהבנאים שלו.

גישה למתודות של שדות של מחלקות לא נגישות

- האם הטיפוס הדינמי יכול להיות ממחלקה לא נגישה?
  - כן. אם הטיפוס הסטטי הוא ממנשק/מחלקה נגיש/ה (ראה דוגמא).
- package a;
  class A{ public int x; }
- האם ניתן לגשת לשדה x של אובייקט מסוג A מחוץ לחבילה a?
- האם ההגדרה בתור public מיותרת?
  - לא, אם בחבילה a נמצאת מחלקה כזו:
    public class AA extends A{ }

גישה למתודות של מחלקות לא נגישות

```
public interface I{  public void f();  }

public class A4 {
   private class A44 implements I{
        public void f() { System.out.println("A44");}
   }
   public I getI(){  return new A44(); }
}

public class B4{
   public static void main(String[] args){
        I o = new A4().getI();
        o.f();
   }
}
```

הטיפוס הדינמי יכול להיות לא נגיש

מנשקים

- המתודות במנשק הן תמיד public abstract (גם אם ה-modifiers הושמטו)
- המשתנים במנשק הם תמיד public static final (גם אם ה-modifiers הושמטו)
- מימוש של מנשק יכול להיחשב לירושה המוגבלת למתודות אבסטרקטיות ושדות סטטיים קבועים.

```
public interface I2{
   public void f(String o);
}

public class A7 implements I2{
   public void f(Object o){}
}
```

Compilation error: A7 does not OVERRIDE I2.f(String) ➔ it should be declared as abstract

Does this compile?

Overloading

```
public class A8 {
    public void f(Object o){System.out.println("f(Object)");}
    public void f(String s){System.out.println("f(String)");}
    public void f(Object o, String s)
                {System.out.println("f(Object,String)");}
    public void f(String s, Object o)
                {System.out.println("f(String,Object)");}
    public static void main(String[] args) {
        A8 a = new A8();
        a.f(null);
        a.f(null,null);
    }
}
```

`f(String)` - **most specific** method

•Which method is called?
•Is this call ok?

compilation error: ambiguous invocation

מחלקות מקוננות

- מחלקה מקוננת – הגדרתה מוכלת בתוך הגדרה של מחלקה אחרת:
  - מחלקה מקוננת סטטית
  - מחלקה פנימית
  - מחלקה מקומית
  - מחלקה אנונימית
- האם יש גישה ממחלקה מקוננת לשדות/מתודות פרטיים של מחלקה עוטפת?
  - כן
- האם למחלקה יש גישה לשדות/מתודות פרטיים של מחלקה מקוננת?
  - כן – למעט מחלקות מקוננות אנונימיות.
- לאילו משתנים יש גישה ממחלקות מקומיות?
  - משתני מחלקה ומופע* של כל המחלקות העוטפות (*אם המתודה העוטפת אינה סטטית) ומשתנים לוקליים קבועים (final) ב- scope.