

## ירושה (Inheritance)

- מספקת שימוש חוזר ויכולת הרחבה
- ניתן להגדיר בדרך זו מחלקה על בסיס ישויות אחרות במערכת:
  - מחלקה אחרת (בדיוק אחת) ו / או
  - מנשקים אחרים (0 או יותר)
- דוגמא: צורות גיאומטריות במישור

## תוכנה 1 בשפת Java: ירושה ופולימורפיזם

תרגול מספר 9  
אורנית דרור ואוהד ברזילי  
אוניברסיטת תל אביב

## class Polygon

```
/** @inv vertices.size() >=3,
 * "A polygon consists of at least 3 points"
 */
public class Polygon {
    /** @pre vertices.size() >=3,
     * "A polygon consists of at least 3 points"
     */
    public Polygon(List<Point> vertices) {
        this.vertices = vertices;
    }
    ...
    /** Successive points making up the polygon */
    private List<Point> vertices;
}
```

## class Polygon

```
Polygon
public Polygon(List<Point> vertices)
public double perimeter()
public void display()
public void rotate(Point center, double angle)
public void translate(Point p)
public int count()
...
```

## דוגמא למימוש מתודה

```
/**
 * Rotate by angle around center
 */
public void rotate(Point center, double angle) {
    for (Point p : vertices) {
        p.rotate(center, angle);
    }
}
```

## class Polygon

```
public class Polygon {
    ...
    /** Length of the perimeter */
    public double perimeter() { ... }
    /** Display polygon on screen */
    public void display() { ... }
    /** Rotate by angle around center */
    public void rotate(Point center, double angle) { ... }
    /**
     * Move by p.x() horizontally and p.y() vertically
     */
    public void translate(Point p) { ... }
    /** Returns the number of sides */
    public int count() { return vertices.size(); }
}
```

## class Parallelogram

○ מקבילית היא סוג של פוליגון  
 ○ Parallelogram היא מקרה פרטי של Polygon

```

class Polygon {
public double perimeter()
public void rotate(Point center, double angle)
public void translate(Point p)
public int count()
...
}
    
```

```

class Parallelogram extends Polygon {
...
}
    
```

קשר ירשה  
 ב-JAVA הרחבה (extension)

כל תכונות המצולע (המתמדות)  
 תקפות למקבילית

8

## דוגמא למימוש מתודה

```

/** Returns the polygon's perimeter */
public double perimeter() {
double result = 0.0;

for (int i = 0 ; i < count() ; i++) {
Point curr = verices.get(i);
Point next = verices.get((i+1) % count());
result += curr.distance(next);
}

return result;
}
    
```

תוכנה 1 בשפת JAVA  
 ארנית דרור ואוהד ברזילי

7

## הבנאי

```

public class Parallelogram extends Polygon {
/**
 * @pre vertices.size() == 4, "exactly 4 points"
 * @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<Point> vertices) {
super(vertices);
// Setting sidel, side2
...
}
...
}
    
```

תוכנה 1 בשפת JAVA  
 ארנית דרור ואוהד ברזילי

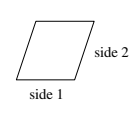
10

## class Parallelogram

```

/**
 * @inv size() == 4 ,
 * "A parallelogram has exactly 4 points"
 * @imp_inv
 * vertices.get(0).distance(vertices.get(1)) ==
 * sidel
 * Similar invariants for the other 3 sides.
 */
public class Parallelogram extends Polygon {
...

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



תוכנה 1 בשפת JAVA  
 ארנית דרור ואוהד ברזילי

9

## מונחי ירשה

```

class Polygon {
public Polygon(List<Point> vertices)
public double perimeter()
public void display()
public void rotate(Point center, double angle)
public void translate(Point p)
public int count()
...
}
    
```

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

```

class Parallelogram {
public Parallelogram(List<Point> vertices)
public double getSidel()
public double getSize2();
...
}
    
```

קשר ירשה  
 ב-JAVA הרחבה (extension)

יצא  
 מחלקה נגזרת (derived)  
 תת מחלקה (subclass)

12

## Overriding perimeter()

```

public class Parallelogram extends Polygon {
...

/** Returns the polygon's perimeter */
public double perimeter() {
return 2 * (sidel + side2);
}

...
}
    
```

תוכנה 1 בשפת JAVA  
 ארנית דרור ואוהד ברזילי

11

## private VS. protected

### בעד protected

- `Rectangle is a Polygon`, הוא עומד ב"מבחן ההחלפה" ולכן לא הגיוני שלא יהיו לו אותן הזכויות.
- `Rectangle has a Polygon` (מכיל בתוכו) ולכן יש צורך לאפשר לו גישה יעילה ופשוטה למימוש הפנימי

## private VS. protected

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

## 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

## private vs. protected

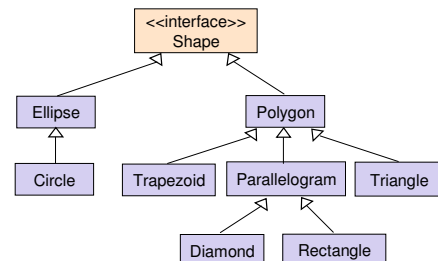
### בעד private:

- 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

## רב צורתיות (Polymorphism)

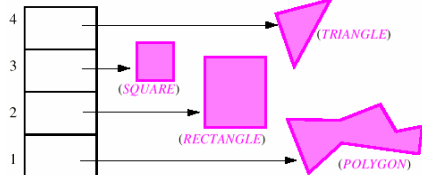
- היכולת של הפנייה (reference) להתייחס בזמן ריצה לעצמים ממחלקות שונות
- תכונה זו מושגת בעזרת ירושה
- העצם המוצבע אינו משנה את טיפוסו. הפנייה מטיפוס מסוים עשויה להצביע בפועל לטיפוס ממחלקה נגזרת

## היררכית מחלקות ומנשקים



## מבנה נתונים פולימורפי

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



20

## דוגמא לפולימורפיזם

```
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;
}
```

תוכנה 1 בשפת JAVA  
אורנית דרור ואוהד ברזילי

19

## 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);
}
```

תוכנה 1 בשפת JAVA  
אורנית דרור ואוהד ברזילי

22

## 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 המוצג

תוכנה 1 בשפת JAVA  
אורנית דרור ואוהד ברזילי

21

## זימון מתודה - דוגמא

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

תוכנה 1 בשפת JAVA  
אורנית דרור ואוהד ברזילי

24

## 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
}
```

תוכנה 1 בשפת JAVA  
אורנית דרור ואוהד ברזילי

23

## 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

26

## 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 הפניה
  - הטיפוס הדינמי: טיפוס העצם המוצבע
  - הטיפוס הדינמי חייב להיות נגזרת של הטיפוס הסטטי

```

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

טיפוס הפניה של הפניה

טיפוס העצם

הטיפוס הדינמי של הפניה

המכון לביטחון ופרטיות  
אוניברסיטת בר-אילן

25

## ירושה וטענות (assertions)

- תנאי קדם, תנאי בתר ושמורות שהוגדרו עבור מחלקה תקפים גם לגבי צאצאיה, ועשויים להשתנות
- עצם ממחלקה נגזרת המוצבע ע"י עצם (מצביע או הפנייה) מטיפוס מחלקת הבסיס צריך לקיים את שמורת מחלקת הבסיס
- מכאן ששמורה של כל מחלקה יכולה להיות שונה או חזקה יותר משמורת הוריה

המכון לביטחון ופרטיות  
אוניברסיטת בר-אילן

28

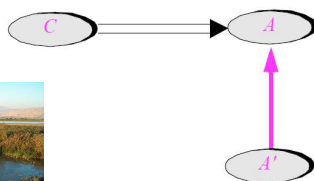
## ירושה וחוזים

דוגמאות נוספות

27

## קבלנות משנה - השמורה

- בפועל, המצביע ל-A מצביע ל-A', מחלקה הנורשת מ-A
- ברור שכדי לקיים יחסים פולימורפים תקינים על A' לקיים לפחות את שמורת A



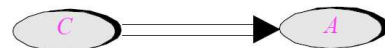
30

## קבלנות משנה



- מחלקת C היא לקוחה של מחלקה A, כלומר:
  - יש ל-C הפנייה ל-A (אחד השדות)
  - או
  - אחת המתודות של C מקבלת פרמטר מטיפוס A (הפנייה ל-A)

- C מכירה את השמורה של A ומצפה מ-A לקיים אותה



המכון לביטחון ופרטיות  
אוניברסיטת בר-אילן

29

## דוגמא

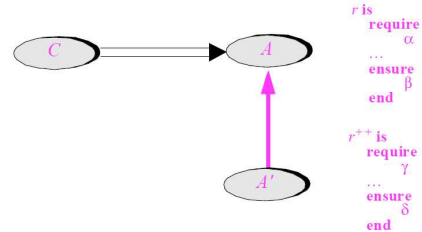
□ בתוך המחלקה Client מופיע הקוד הבא:

```
public class Client {
    ...
    public static void g(String args[])
    {
        List<String> l = Arrays.asList(args);
        ...
    }
}
```

- בדוגמא זו Client הוא הלקוח (C) ו- List הוא הספק (A)
- ואולם ברור ש- 1 מצביע בפועל לעצם ממחלקה שממשת את List (אולי ArrayList). מחלקה זו היא קבלנית משנה (A')
- הלקוח, שאינו מכיר את קבלן המשנה שלו, מצפה ממנו לעמוד בחוזה המקורי (החוזה מול הספק)

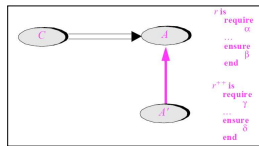
## קבלנות משנה – תנאי קדם ובתר

- המחלקה A' מסתירה (overrides) רוטינה של A
- מה יש לדרוש מתנאי הקדם והבתר של המתודה החדשה ביחס לאלו של הרוטינה המקורית?



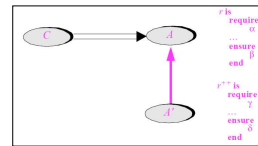
## קבלנות משנה – תנאי בתר

- משיקולים דומים על תנאי הבתר של המחלקה הנגזרת להיות שווה או חזק יותר מתנאי הבתר המקורי
- ללקוח C 'הבוטח'  $\beta$  ע"י A ואסור שמאחורי הקלעים יסופק  $\delta$  החלש ממנו
- מנגנון זה מכונה "קבלנות משנה" (subcontracting)



## קבלנות משנה – תנאי קדם

- נתבונן בקריאה  $r()$  המופיעה במחלקה C
- על C לקיים את תנאי הקדם של  $A.r()$ , היא כלל אינה מכירה את המחלקה A' ואינה יודעת על קיום  $A'.r()$
- לכן על תנאי הקדם המוגדר במחלקה הנגזרת להיות שווה או חלש יותר מתנאי הקדם המקורי



## תנאי קדם אפקטיבי

- תנאי הקדם 'אמיתי' של מתודה שהוגדרה מחדש במחלקה כלשהי, הוא ה-OR הלוגי של כל תנאי הקדם של מתודה זו בכל הוריה של אותה מחלקה לאורך עץ הירושה
- אם עבור רמה (מחלקה) מסוימת בעץ הירושה לא הוגדר תנאי קדם למתודה זו, ניתן להתייחס לתנאי הקדם שם כ- FALSE
- כותב תנאי הקדם של המתודה שהוגדרה מחדש במחלקה כלשהי, יכול להגדיר אותו בצורה מרומזת (implicit) ע"י ציון הטענות החדשות בלבד

## השמורה האפקטיבית

- השמורה 'אמיתית' של מחלקה מורכבת מ-AND לוגי של כל הטענות המופיעות בשמורת אותה מחלקה ובכל הוריה לאורך עץ הירושה
- אם עבור רמה (מחלקה) מסוימת בעץ הירושה לא הוגדרה שמורה, ניתן להתייחס לשמורה שלה כ- TRUE
- כותב מחלקה יכול להגדיר את השמורה שלה בצורה מרומזת (implicit) ע"י ציון הטענות החדשות בלבד

## דוגמא

```
public class MATRIX {
    ...
    /** inverse of current with precision epsilon
     * @pre epsilon >= 10 ^(-6)
     * @post (this.mult($prev(this)) - ONE).norm <= epsilon
     */
    void invert(double epsilon);
    ...
}
```



תוכנה 1 בשפת Java  
אנדרית דרור ואנדר ברזילי

38

## תנאי בתר אפקטיבי

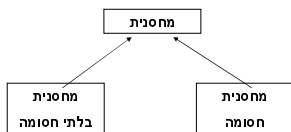
- תנאי הבתר האמיתי של מתודה שהוגדרה מחדש במחלקה כלשהי הוא ה AND הלוגי של כל תנאי הבתר של מתודה זו בכל הוריה של אותה מחלקה לאורך עץ הירושה
- אם עבור רמה (מחלקה) מסוימת בעץ הירושה לא הוגדר תנאי קדם למתודה זו, ניתן להתייחס לתנאי הקדם שם כ- TRUE
- כותב תנאי הבתר של המתודה שהוגדרה מחדש במחלקה כלשהי יכול להגדיר אותו בצורה מרומזת (implicit) ע"י ציון הטענות החדשות בלבד

תוכנה 1 בשפת Java  
אנדרית דרור ואנדר ברזילי

37

## תנאי קדם מופשט

- מהי ההיררכיה בין 3 המחלקות: מחסנית, מחסנית חסומה, מחסנית בלתי חסומה?



- מה יהיה תנאי הקדם של המתודה put במחלקה מחסנית?

תוכנה 1 בשפת Java  
אנדרית דרור ואנדר ברזילי

40

## דוגמא



```
public class ACCURATE_MATRIX extends MATRIX {
    ...
    /** inverse of current with precision epsilon
     * @pre epsilon >= 10^(-20)
     * @post (this.mult($prev(this)) - ONE).norm <= epsilon/2
     */
    void invert(double epsilon);
    ...
}
```

- בשפת Eiffel כדי להדגיש שהחזרה של מתודה שהוגדרה מחדש אינו עומד בפני עצמו אלא תלוי בהיררכיה החליפו את התגיות ensure then -i require else -i require בהתאמה

תוכנה 1 בשפת Java  
אנדרית דרור ואנדר ברזילי

39

## תנאי קדם מופשט

- כאשר מחלקת הבסיס מופשטת, תנאי קדם טריויאליים מחייבים לפעמים ראייה לעתיד, כדי שלא יחזקו במחלקות נגזרת
- ראייה לעתיד אינה דבר מופרך במחלקות מופשטות
- נתבונן בדוגמא נוספת: מערכת תוכנה אשר מיוצגים בה כלי תחבורה שונים כגון מכונית, אווירון ואופניים

תוכנה 1 בשפת Java  
אנדרית דרור ואנדר ברזילי

42

## תנאי קדם מופשט

- תנאי הקדם לא יכול להיות ריק (TRUE) כי אז הוא יחזק ע"י המחסנית החסומה
- תנאי הקדם צריך להיות full()! כאשר full() היא מתודה המחזירה תמיד false, שתוגדר מחדש במחלקה מחסנית חסומה להחזיר count()==capacity()
- תנאי קדם המכיל מתודות שנדרסות במורד הירושה נקרא תנאי קדם מופשט
- למרות שתנאי הקדם הקונקרטי אכן מתחזק ע"י המחסנית החסומה תנאי הקדם המופשט נשאר ללא שינוי

תוכנה 1 בשפת Java  
אנדרית דרור ואנדר ברזילי

41

### דוגמא

```

classDiagram
    class VEHICLE
    class CAR
    class BICYCLE
    VEHICLE <|-- CAR
    VEHICLE <|-- BICYCLE
    
```

- מהו תנאי הקדם של המתודה go() של המחלקה VEHICLE?
- על פניו – אין כל תנאי קדם לפעולה מופשטת
- מה עם המחלקה CAR? – לה בטח יש דרישות כגון hasFuel()
- מה עם המחלקה BICYCLE? – לה בטח יש דרישות כגון hasAir()
- איך VEHICLE תגדיר תנאי קדם ל go() גם כללי מספיק וגם שלא יחזק ע"י אף אחד מירשותיה?

תוכנה 1 בשפת Java  
אוריית דרור ואחוד ברזילי

44

### ראייה לטווח רחוק

- האבולוציה של היררכית מחלקות כלי הרכב לא מתחילה בגזירת מחלקות קונקרטיות שיירשו מ VEHICLE
- הגיוני יותר שבמהלך מימוש ולא עיצוב המחלקות CAR ו-AIRPLANE נגלה שיש להן הרבה מן המשותף, וכדי למנוע שכפול קוד ניצור מחלקה שלישית - VEHICLE שתכיל את החיתוך של שתיהן
- אף כלי רכב אינו רק VEHICLE
- בראייה זו, לא מוגזם לדרוש ממחלקה מופשטת ניסוח תנאי קדם מופשט

תוכנה 1 בשפת Java  
אוריית דרור ואחוד ברזילי

43

### מתי לרשת? - דיון

- המחלקה CAR\_OWNER עשויה לרשת מ PERSON אבל עדיף שתהיה לקוחה של CAR
- יחס is-a לעומת is-part-of on
- פרט למנגנון הרב-צורתיות (polymorphism) ירושה לעולם אינה הכרחית
- במקום ש B יירש מ-A, ל-B יכולה להיות התכונה A (שדה מוכל או מצביע)
- To be is also to have אבל לא להיפך (משאית היא מכונית כלומר חלק בה הוא מכונית)
- לפעמים נוח לשאול "האם יכולים להיות לו שניים?"
- לדוגמא: למכונית יש מנוע

תוכנה 1 בשפת Java  
אוריית דרור ואחוד ברזילי

46

### פתרון

```

classDiagram
    class VEHICLE
    class CAR
    class BICYCLE
    VEHICLE <|-- CAR
    VEHICLE <|-- BICYCLE
    
```

- מתודה בולאנית כגון canGo() תעשה את העבודה
- המתודה תוגדר כמחזירה TRUE עבור VEHICLE (או שתוגדר כ abstract), ועבור כל אחת מירשותיה תוגדר לפי המחלקה האמורה
- בעצם המתודה go() היתה צריכה להיקרא "go\_because\_you\_can()" וכך לא היתה כל הפתעה בתנאי הקדם "המוזר"

תוכנה 1 בשפת Java  
אוריית דרור ואחוד ברזילי

45

### שימוש חוזר במנשק ובמימוש

יחס ירושה	יחס שימוש (client-supplier)
שימוש חוזר דרך מימוש	שימוש חוזר דרך מנשק
הסתרת מידע מוגבלת	יש הסתרת מידע
הגנה מוגבלת מפני שינויים	הגנה מפני שינויים במימוש המקורי

תוכנה 1 בשפת Java  
אוריית דרור ואחוד ברזילי

47