

ירושה ממחלקות קיימות

- ראינו בהרצאה שתי דרכים לשימוש חוזר בקוד של מחלקה קיימת:
 - הכלה + האצלה
 - ירושה
- המחלקה היורשת יכולה להוסיף פונקציונליות שלא היתה קיימת במחלקת הבסיס, או לשנות פונקציונליות שקיבלה בירושה
- בדוגמא הבאה אנו יורשים מהמחלקה Turtle שכבר הכרנו ומוסיפים לה פונקציונליות חדשה: drawSquare

2

תוכנה 1

תרגול מספר 9: הורשה

בית הספר למדעי המחשב
אוניברסיטת תל אביב

דריסת שורות

- המחלקה היורשת בדרך כלל מבטאת תת-משפחה של העצמים ממחלקת הבסיס
- המחלקה היורשת יכולה לדרוס שירותים שהתקבלו בירושה
- כדי להשתמש בשירות המקורי (למשל ע"י השרות הדורס בעצמו) ניתן לפנות לשירות בתחביר: `super.methodName(...)`
- בדוגמא הבאה אנו מגדירים **צב שיכור** הירש מהמחלקה Turtle ודורס את השרות `moveForward`

4

צב חכם

```
/**
 * A logo turtle that knows how to draw squares
 */
public class SmartTurtle extends Turtle {
    public void drawSquare(int edge) {
        for (int i = 0; i < 4; i++) {
            moveForward(edge);
            turnLeft(90);
        }
    }
}
```

ירושה ממחלקה קיימת

הוספת שרות חדש

שימוש בשירותים ממחלקת האם

3

נראות והורשה

- שדות ושירותים פרטיים (private) של מחלקת הבסיס אינם נגישים למחלקה היורשת
- כדי לאפשר גישה למחלקות יורשות יש להגדיר להם נראות **protected**
- שימוש בירושה יעשה בזהירות מרבית, בפרט הרשאות גישה למימוש
- נשתמש ב `protected` רק כאשר אנחנו מתכננים היררכיות ירושה שלמות ושולטים במחלקה היורשת

6

צב שיכור

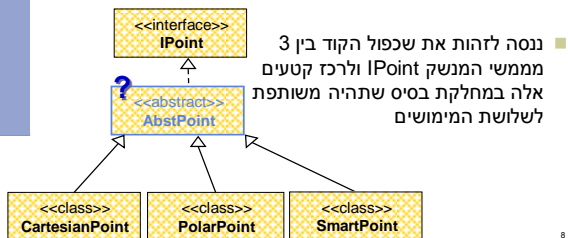
```
/**
 * A drunk turtle is a turtle that "staggers" as it moves forward
 */
public class DrunkTurtle extends Turtle {
    /**
     * Zigzag forward a specified number of units. At each step the turtle may
     * make a turn of up to 30 degrees.
     *
     * @param units
     *        - number of steps to take
     */
    @Override
    public void moveForward(double units) {
        for (int i = 0; i < units; i++) {
            if (Math.random() < 0.1) {
                turnLeft((int) (Math.random() * 60 - 30));
            }
            super.moveForward(1);
        }
    }
}
```

דריסה של שירות קיים

5

צד הספק

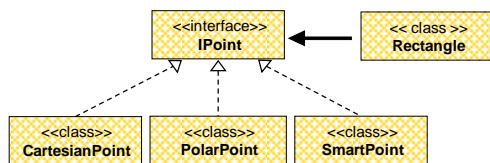
- לעומת זאת, מנגנון ההורשה חוסך שכפול קוד בצד הספק
- ע"י הורשה מקבלת מחלקה את קטע הקוד בירושה במקום לחזור עליו. שני הספקים חולקים אותו הקוד



נסה לזהות את שכפול הקוד בין 3 ממשי הממשק IPoint ולרכז קטעים אלה במחלקת בסיס שתהיה משותפת לשלושת המימושים

צד הלקוח

- בהרצאה ראינו את הממשק IPoint, והצגנו 3 מימושים שונים עבורו
- ראינו כי **לקוחות** התלויים בממשק IPoint בלבד, ולא מכירים את המחלקות המממשות **אדישים** לשינויים עתידיים בקוד הספק
- שימוש **בממשקים** חוסך שכפול קוד לקוח, בכך שאותו קטע קוד עובד בצורה נכונה עם מגוון ספקים (פולימורפיזם)



מחלקות מופשטות - דוגמה

```

public abstract class A {
    public void f() {
        System.out.println("A.f!");
    }

    abstract public void g();
}

A a = new A();

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

A a = new B();
    
```

מחלקה פשוטה:



מחלקות מופשטות

- מחלקה מופשטת מוגדרת ע"י המלה השמורה **abstract**
- לא ניתן ליצור מופע של מחלקה מופשטת (בדומה לממשק)
- יכולה לממש ממשק מבלי לממש את כל השירותים המוגדרים בו
- זהו מנגנון מועיל להימנע משכפול קוד במחלקות יורשות



CartesianPoint

```

public void rotate(double angle) {
    double currentTheta = Math.atan2(y,x);
    double currentRho = rho();

    x = currentRho *
    Math.cos(currentTheta+angle);
    y = currentRho * Math.sin(currentTheta+angle);
}

public void translate(double dx, double dy) {
    x += dx;
    y += dy;
}
    
```

PolarPoint

```

public void rotate(double angle) {
    theta += angle;
}

public void translate(double dx, double dy) {
    double newX = x() + dx;
    double newY = y() + dy;
    r = Math.sqrt(newX*newX + newY*newY);
    theta = Math.atan2(newY,newX);
}
    
```

גם כאן קשה לראות דמיון בין מימושי המתודות, למימושים קשר הדוק לייצוג שנבחר לשדות

CartesianPoint

```

private double x;
private double y;

public CartesianPoint(double x, double y) {
    this.x = x;
    this.y = y;
}

public double x() { return x; }
public double y() { return y; }
public double rho() { return Math.sqrt(x*x + y*y); }
public double theta() { return Math.atan2(y,x); }
    
```

PolarPoint

```

private double r;
private double theta;

public PolarPoint(double r, double theta) {
    this.r = r;
    this.theta = theta;
}

public double x() { return r * Math.cos(theta); }
public double y() { return r * Math.sin(theta); }
public double rho() { return r; }
public double theta() { return theta; }
    
```

קשה לראות דמיון בין מימושי המתודות במקרה זה. כל 4 המתודות בסיסיות ויש להן קשר הדוק לייצוג שנבחר לשדות

CartesianPoint	PolarPoint
<pre>public double distance(IPoint other) { double deltaX = x - other.x(); double deltaY = y - other.y(); return Math.sqrt((x - other.x()) * (x - other.x()) + (y - other.y()) * (y - other.y())); }</pre>	<pre>public double distance(IPoint other) { double deltaX = x() - other.x(); double deltaY = y() - other.y(); return Math.sqrt(deltaX * deltaX + deltaY * deltaY); }</pre>

14

CartesianPoint	PolarPoint
<pre>public double distance(IPoint other) { return Math.sqrt((x - other.x()) * (x - other.x()) + (y - other.y()) * (y - other.y())); }</pre>	<pre>public double distance(IPoint other) { double deltaX = x() - other.x(); double deltaY = y() - other.y(); return Math.sqrt(deltaX * deltaX + deltaY * deltaY); }</pre>

הקוד דומה אבל לא זהה, נראה מה ניתן לעשות...

נוסח לשכתב את CartesianPoint ע"י הוספת משתני העזר deltaX ו-deltaY

13

CartesianPoint	PolarPoint
<pre>public double distance(IPoint other) { double deltaX = x() - other.x(); double deltaY = y() - other.y(); return Math.sqrt(deltaX * deltaX + (deltaY * deltaY)); }</pre>	<pre>public double distance(IPoint other) { double deltaX = x() - other.x(); double deltaY = y() - other.y(); return Math.sqrt(deltaX * deltaX + (deltaY * deltaY)); }</pre>

שתי המתודות זהות לחלוטין!
ניתן להעביר את המתודה למחלקה AbstPoint
ולמחוק אותה מהמחלקות CartesianPoint ו-PolarPoint

16

CartesianPoint	PolarPoint
<pre>public double distance(IPoint other) { double deltaX = x - other.x(); double deltaY = y - other.y(); return Math.sqrt(deltaX * deltaX + (deltaY * deltaY)); }</pre>	<pre>public double distance(IPoint other) { double deltaX = x() - other.x(); double deltaY = y() - other.y(); return Math.sqrt(deltaX * deltaX + deltaY * deltaY); }</pre>

נשאר הכול אחד

נחליף את x להיות x() -
במאזן ביצועים לעמדת כלליות נעדיף תמיד את הכלליות

15

אתחול עכשיו

CartesianPoint	PolarPoint
<pre>public String toString() { return "x=" + x + ", y=" + y + ", r=" + rho() + ", theta=" + theta() + "°"; }</pre>	<pre>public String toString() { return "x=" + x() + ", y=" + y() + ", r=" + r + ", theta=" + theta + "°"; }</pre>

תהליך דומה ניתן גם לבצע עבור עובר toString

17

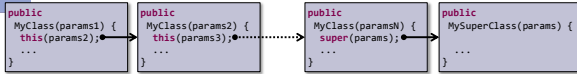
אתחולים ובנאים

- יצירת מופע חדש של עצם כוללת: הקצאת זכרון, אתחול, הפעלת בנאים והשמה לשדות
- במסגרת ריצת הבנאי נקראים גם הבנאים של מחלקת הבסיס
- תהליך זה מבלבל כי לשדה מסוים ניתן לבצע השמות גם ע"י אתחול, וגם ע"י מספר בנאים (אחרון קובע)
- בשקפים הבאים נתאר במדויק את התהליך
- נעזר בדוגמא

19

תזכורת

- בשורה הראשונה של כל בנאי חייבים לקרוא לאחד משניים:**
 - בנאי של מחלקת האב (`super(...)`)
 - אם לא נכתבת קריאה מפורשת, נקרא בנאי ברירת המחדל (`super()`)
 - אם אין כזה, תהיה שגיאה!
- כאשר יש **העמסת בנאים** – לבנאי אחר בעזרת (`this(...)`)
- בסופו של דבר נגיע לבנאי שקורא ל- (`super(...)`)



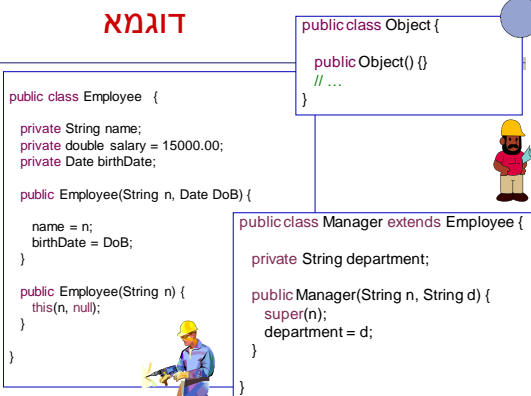
20

מה הסדר ביצירת מופע של מחלקה?

- שלב ראשון:** הקצאת זיכרון לשדות העצם והצבת ערכי ברירת מחדל
- שלב שני:** נקרא הבנאי (לפי חתימת `new`) והאלגוריתם הבא מופעל:
 - Bind constructor parameters.
 - If explicit `this()`, call recursively, and then skip to Step 5.
 - Call recursively the implicit or explicit `super(...)` [except for `Object` because `Object` has no parent class]
 - Execute the explicit field initializers.
 - Execute the body of the current constructor.

21

דוגמא



תמונת הזכרון

- Basic initialization**
 - Allocate memory for the complete `Manager` object
 - Initialize all fields to their default values
- Call constructor: `Manager("Joe Smith", "Sales")`**
 - Bind constructor parameters: `n="Joe Smith", d="Sales"`
 - No explicit `this()` call
 - Call `super(n)` for `Employee(String)`
 - Bind constructor parameters: `n="Joe Smith", DoB=null`
 - No explicit `this()` call
 - Call `super()` for `Object()`
 - No `this()` call
 - No `super()` call (`Object` is the root)
 - No explicit field initialization for `Object`
 - No method body to call
 - Initialize explicit `Employee` fields: `salary=15000.00;`
 - Execute body: `name="Joe Smith", date=null;`
 - Steps 3-4 skipped
 - Execute body: No body in `Employee(String)`
 - No explicit initializers for `Manager`
 - Execute body: `department="Sales"`

(String) Name	"Joe Smith"
(double) Salary	15000.0
(Date) Birth Date	null
(String) Department	"Sales"

```

public class Employee extends Object {
    private String name;
    private double salary = 15000.00;
    private Date birthDate;

    public Employee(String n, Date DoB) {
        // implicit super();
        name = n;
        birthDate = DoB;
    }

    public Employee(String n) {
        this(n, null);
    }
}
    
```

```

public class Manager
    extends Employee {
    private String department;
    public Manager(String n, String d) {
        super(n);
        department = d;
    }
}
    
```

הרצת הדוגמא

- מה קורה כאשר ה `JVM` מריץ את השורה `Manager m = new Manager("Joe Smith", "Sales");`
- שלב ראשון: הקצאת זיכרון לשדות העצם והצבת ערכי ברירת מחדל



```

(String)Name
(double)Salary
(Date)Birth Date
(String)Department
    
```

23