

## 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 שראינו בתחילת הסמסטר, ומוסיפים לה פונקציונליות חדשה: drawSquare

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

2

## תוכנה 1 בשפת Java

תרגול מספר 8: הורשה  
נעמה מאיר ומתי שמרת

בית הספר למדעי המחשב  
אוניברסיטת תל אביב

## דריסת שרותים

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

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

4

## צב חכם

```
/**
 * A logo turtle that knows how to draw squares
 */
class SmartTurtle extends Turtle {

    /**
     * Draws a square
     * @param edge the size of the square edge
     */
    public void drawSquare(int edge) {
        for(int i=0; i<4; i=i+1) {
            moveForward(edge);
            turnLeft(90);
        }
    }
}
```

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

3

## נראות והורשה

- שדות ושרותים פרטיים (private) של מחלקת הבסיס אינם נגישים למחלקה היורשת
- כדי לאפשר גישה למחלקות יורשות יש להגדיר להם נראות `protected`
- למשל, אם במחלקה Turtle מצב הזנב הוא שדה פרטי ואין שאילתות על מצב הזנב, צריך יורש המחלקה לממש פונקציונליות זו מחדש
- איך נממש צב אשר מצייר קו מרוסק במקום קו מלא?

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

6

## צב שיכור

```
/**
 * A Drunk turtle is a turtle. a drunk turtle can't move
 * Forward in a straight line but instead it moves
 * in a zig zag fashion
 */
public class DrunkTurtle extends Turtle {

    /**
     * Advances the turtle forwards by a given number of
     * steps. each steps will consist of up to 4 pixels and
     * will follow a left turn of (-30,30) degrees.
     * @param steps The number of steps the turtle should
     * advance by.
     */
    public void moveForward(double steps) {
        for(int i = 0; i < steps; i++) {
            if (Math.random() < 0.1) {
                turnLeft((int)(Math.random() * 60) - 30);
            }
            super.moveForward(1);
        }
    }
}
```

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

5

```

/**
 * Advances the turtle forwards by a given number of
 * units, if its tail is down it will
 * draw a dashed line when moving.
 * @param units The number of units the turtle should
 * advance by.
 */
public void moveForward(double units) {
    if(!down)
        super.moveForward(units); //not drawing
    else {
        int times = (int)units/10;
        int leftOver = (int)units%10;
        //creating the lines each of size 10
        for(int i=0; i<times; i=i+1) {
            super.moveForward(10);
            changeTail();
        }
        super.moveForward(leftOver);
        tailDown();
    }
}

//changing the tail position
private void changeTail() {
    if(down)
        tailUp();
    else
        tailDown();
}
}

```

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

```

/**
 * A lined turtle is a turtle that draws dashed line
 * when moving forward.
 */
public class LinedTurtle extends SmartTurtle {
    //Keeps the tail position
    private boolean down;

    /**
     * Constructs a new lined turtle.
     */
    public LinedTurtle() {
        down = false;
    }

    /**
     * Lowers the tail of the turtle. Consequent movements
     * of the turtle will leave behind it a dashed line.
     */
    public void tailDown() {
        down = true;
        super.tailDown();
    }

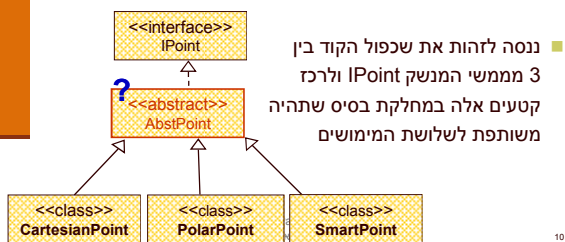
    /**
     * Raises the tail of the turtle. Consequent movements
     * of the turtle will not leave any mark behind it.
     */
    public void tailUp() {
        down = false;
        super.tailUp();
    }
}

```

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

## צד הספק

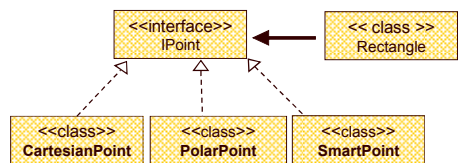
- לעומת זאת, מנגנון ההורשה חוסך שכול קוד בצד הספק
- ע"י הורשה מקבלת מחלקה את קטע הקוד בירושה במקום לחזור עליו. שני הספקים חולקים אותו הקוד



נסה לזהות את שכול 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 שכול 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:



תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

## מחלקות מופשטות



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

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

CartesianPoint	PolarPoint
<pre>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; }</pre>	<pre>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); }</pre>

גם כאן קשה לראות דמיון בין מימשי המתודות, למימשים קשר הדוק לייצוג שנבחר ל**לשדות**

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

CartesianPoint	PolarPoint
<pre>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); }</pre>	<pre>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; }</pre>

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

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

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>

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

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...  
נסה לשכתב את **CartesianPoint** ע"י הוספת משתני העזר  $\Delta X$  ו- $\Delta Y$

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

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**

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

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 הבדל אחד  
נחליף את  $x$  להיות  $x()$  - במאזן **ביצועים** לעומת **כלליות** נעדיף תמיד את **הכלליות**

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

## 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

## CartesianPoint

```
public String toString(){
    return "x=" + x + ", y=" + y +
        ", r=" + rho() + ", theta=" + theta() + "°";
}
```

## PolarPoint

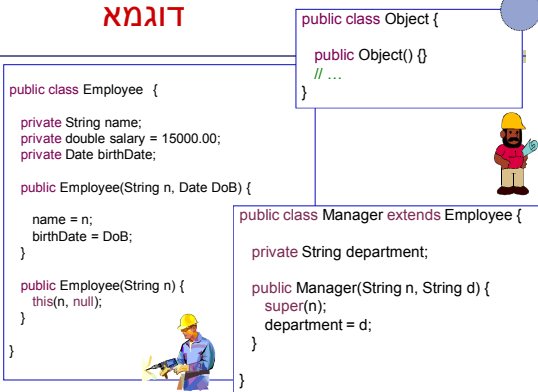
```
public String toString() {
    return "x=" + x0 + ", y=" + y0 +
        ", r=" + r + ", theta=" + theta + "°";
}
```

תהליך דומה ניתן גם לבצע עבור toString

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב

19

## דוגמא



## מה הסדר ביצירת מופע של מחלקה?

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

## תמונת הזכרון

- **Basic initialization**
    - Allocate memory for the complete Manager object
    - Initialize all instance variables 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"
    - Call this(n, null) for Employee(String, Date)
1. Bind parameters.  
2. If explicit this(), goto 5.  
3. super().  
4. explicit var. init.  
5. Execute body
- Bind constructor parameters: n="Joe Smith", DoB=null
  - No explicit this() call
  - Call super() for Object()
    - No binding necessary
    - No this() call
    - No super() call (Object is the root)
    - No explicit variable initialization for Object
  - No method body to call
  - Initialize explicit Employee variables: salary=15000.00;
  - Execute body: name="Joe Smith", date=null;
  - Steps 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
```