

ירושה ממחלקות קיימות

- ראינו בהרצאה שתי דרכים לשימוש חוזר בקוד של מחלקה קיימת:
 - הכלה + האצלה
 - ירושה
- המחלקה הירושת 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

בית הספר למדעי המחשב
אוניברסיטת תל אביב

דריסת שרותים

- המחלקה הירושת בדרך כלל מבטאת תת משפחה של העצמים ממחלקת הבסיס
- המחלקה הירושת 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** הירושה 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 הירושת
- 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 lines
 * when moving forward.
 */
public class LinedTurtle extends BasicTurtle {
    //keeps the tail position.
    private boolean down;

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

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

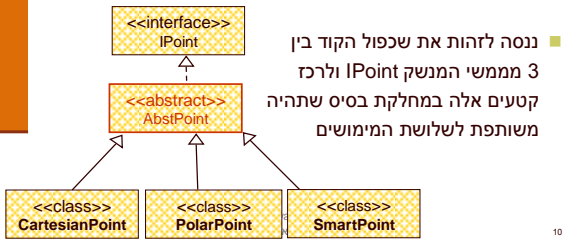
    //keeps the tail of the turtle. Consequent movements
    //of the turtle will not leave any mark behind it.
    public void tailUp() {
        down = false;
        super.tailUp();
    }
}

```

תוכנה 1 בשפת Java
אוניברסיטת תל אביב

צד הספק

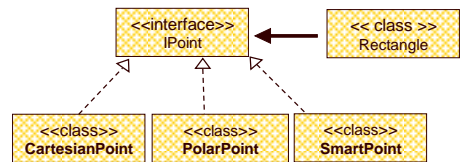
- לעומת זאת, מנגנון ההורשה חוסך שכפול קוד בצד הספק
- ע"י הורשה מקבלת מחלקה את קטע הקוד בירושה במקום לחזור עליו. שני הספקים חולקים אותו הקוד



- ננסה לזהות את שכפול הקוד בין 3 ממשי הממשק IPoint ולרז קטעים אלה במחלקת בסיס שתהיה משותפת לשלושת המימושים

צד הלקוח

- בהרצה 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:



A a = new B();

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 משכפול קוד במחלקות יורשות



תוכנה 1 בשפת Java
אוניברסיטת תל אביב

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);
}
```

תוכנה 1 בשפת Java
אוניברסיטת תל אביב

14

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; }
```

תוכנה 1 בשפת Java
אוניברסיטת תל אביב

13

CartesianPoint

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

PolarPoint

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

תוכנה 1 בשפת Java
אוניברסיטת תל אביב

16

CartesianPoint

```
public double distance(IPoint other) {
    return Math.sqrt((x-other.x()) * (x-other.x()) +
        (y-other.y()) * (y-other.y()));
}
```

PolarPoint

```
public double distance(IPoint other) {
    double deltaX = x()-other.x();
    double deltaY = y()-other.y();

    return Math.sqrt(deltaX*deltaX +
        deltaY*deltaY);
}
```

תוכנה 1 בשפת Java
אוניברסיטת תל אביב

15

אתחולים ובנאים

Extract Superclass Refactoring



- ניתן לבצע תהליך זה בצורה אוטומטית ע"י שכתוב מבני (Refactoring) שנקרא: Extract Superclass
- הגרסה ב-Eclipse עוד לא "מושלמת"

תוכנה 1 בשפת Java
אוניברסיטת תל אביב

17

מה הסדר ביצירת מופע של מחלקה?

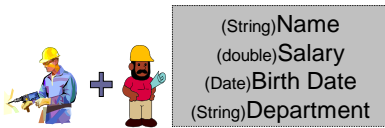
- שלב ראשון:** הקצאת זיכרון לשדות העצם והצבת ערכי ברירת מחדל
- שלב שני:** נקרא הבנאי (לפי חתימת new) והאלגוריתם הבא מופעל:
 - Bind constructor parameters.
 - If explicit this(), call recursively, and then skip to Step 5.
 - Call recursively the implicit or explicit super(...)
 - [except for Object because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

אתחולים ובנאים

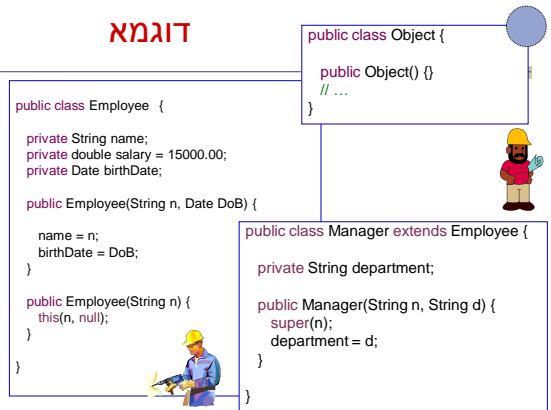
- יצירת מופע חדש של עצם כוללת: הקצאת זכרון, אתחול, הפעלת בנאים והשמה לשדות
- במסגרת ריצת הבנאי נקראים גם הבנאים של מחלקת הבסיס
- תהליך זה מבלבל כי לשדה מסוים ניתן לבצע השמות גם ע"י אתחול, וגם ע"י מספר בנאים (אחרון קובע)
- בשקפים הבאים נתאר במדויק את התהליך
- נעזר בדוגמא

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



דוגמא



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)
 - 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 to the root)
 - No explicit variable initialization for Object
 - No method body to call
- Bind parameters.
 - If explicit this(), goto 5.
 - super(),
 - explicit var. init.
 - Execute body
- 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;
    }
}
```