תוכנה 1 בשפת Java

תרגול מספר 7: הורשה

הדס צור ואסף זריצקי

בית הספר למדעי המחשב
אוניברסיטת תל אביב

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

צב חכם

```
/**
 * A logo turtle that knows how to draw square
 */
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

דריסת שרותים

- המחלקה היורשת בדרך כלל מבטאת תת משפחה של העצמים ממחלקת הבסיס
- המחלקה היורשת יכולה לדרוס שירותים 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

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 stes 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);
        }
    }
}
```

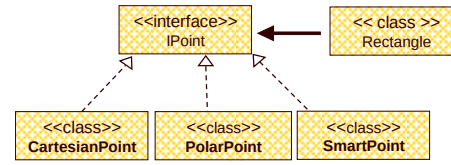
דריסה של שירות קיים

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

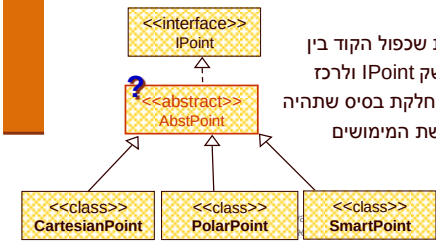
צד הלקוח

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



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 מממשי המנשק IPoint ולרכז קטעים אלה במחלקת בסיס שתהיה משותפת לשלושת המימושים



מחלקות מופשטות

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

מחלקות מופשטות - דוגמא

- מחלקה פשוטה:

```
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();
```

CartesianPoint	PolarPoint
`private double x;` `private double y;`	`private double r;` `private double theta;`

`public CartesianPoint(double x, double y) {`
`this.x = x;`
`this.y = y;`
`}`
	`public PolarPoint(double r, double theta) {`
	`this.r = r;`
	`this.theta = theta;`
	`}`

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

`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 rotate(double angle) {`
	`theta += angle;`
	`}`

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

```
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()));
}
```

PolarPoint

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

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


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);
}
```

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

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

CartesianPoint

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

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

PolarPoint

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

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

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

CartesianPoint

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

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

PolarPoint

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

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

נשאר הבדל אחד

נחליף את x להיות x() –
במאזן **ביצועים** לעומת **כלליות** נעדיף תמיד את **הכלליות**

Method Overloading & Overriding

```
public class A {
   public float foo(float a, float b){…
   }
}

public class B extends A {
   …
}
```

Which of the following methods can be defined in B:

1. float foo(float a, float b){…}
2. ✓ public int foo(int a){…}
3. ✓ public float foo(float p, float q) {…}

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 + ")";
}
```

תהליך דומה ניתן גם לבצע עבור toString

אתחולים ובנאים

- יצירת מופע חדש של עצם כוללת: הקצאת זכרון, אתחול, הפעלת בנאים והשמה לשדות
- במסגרת ריצת הבנאי נקראים גם הבנאי/ים 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

Method Overriding

```
public class A {
  public void print() {
    System.out.println("A");
  }
}

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

```
public class C {
  public static void main(...) {
    B b = new B();
    A a = b;

    b.print();
    a.print();
  }
}
```

The output is: ...ut?
B
B

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.

דוגמא

```
public class Object {

  public Object() {}
  // ...
}
```

```
public class Employee  {

  private String name;
  private double salary = 15000.00;
  private Date birthDate;

  public Employee(String n, Date DoB) {

    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

תמונת הזכרון

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

- **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 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"

1. Bind parameters.
2. If explicit this(), goto 5.
3. super(),
4. explicit var. init.
5. Execute body

```
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;
  }
}
```

Singletone design pattern

- בהקשר של תרגיל 7
- האם צריך לייצר יותר מכלב אחד?
- איך דואגים שיהיה בדיוק instance יחיד?
- Singletone design pattern