**בית הספר למדעי המחשב**
**אוניברסיטת תל אביב**

# תוכנה 1

## תרגול מספר 8:

הורשה

מחלקות אבסטרקטיות

חריגים

# ירושה ממחלקות קיימות

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

- הכלה (aggregation)– במחלקה א' יש שדה מטיפוס מחלקה ב'
- האצלה (delegation)– קוראים מתוך מתודות במחלקה א' למתודות של מחלקה ב'

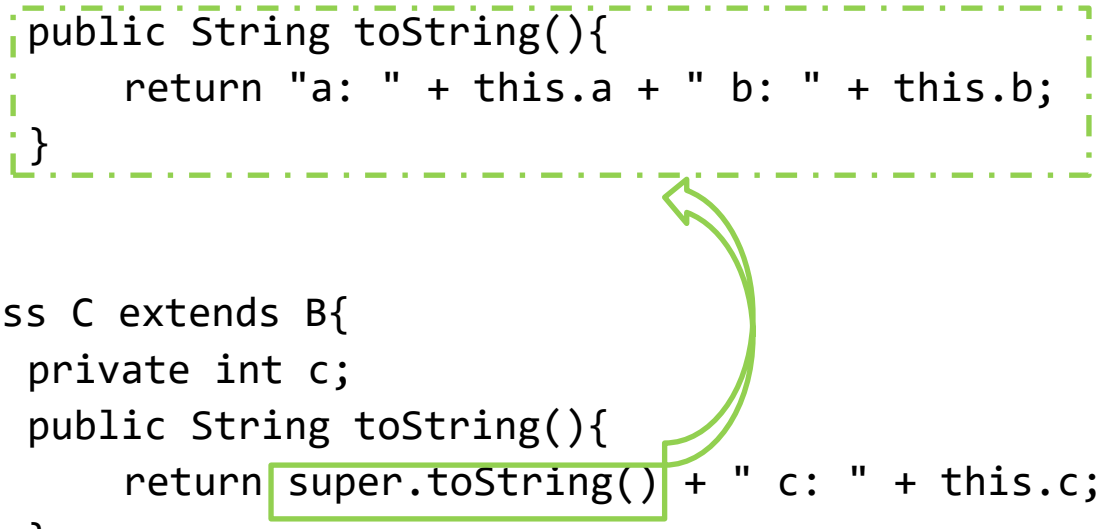
- המחלקה היורשת יכולה **להוסיף** פונקציונאליות שלא היתה קיימת במחלקת הבסיס, או **לשנות** פונקציונאליות שקיבלה בירושה

# דריסת שירותים

- המחלקה היורשת בדרך כלל מייצגת תת-משפחה של מחלקת הבסיס
- המחלקה היורשת יכולה לדרוס שירותים שהתקבלו בירושה
- כדי להשתמש בשירות המקורי (למשל מהשירות הדורס) ניתן לפנות לשירות המקורי בתחביר: super.methodName(…)

# שימוש בשירות המקורי מתוך השירות הדורס

```
class B {
    protected int a;
    protected int b;
    public String toString(){
        return "a: " + this.a + " b: " + this.b;
    }
}

class C extends B{
    private int c;
    public String toString(){
        return super.toString() + " c: " + this.c;
    }
}
```

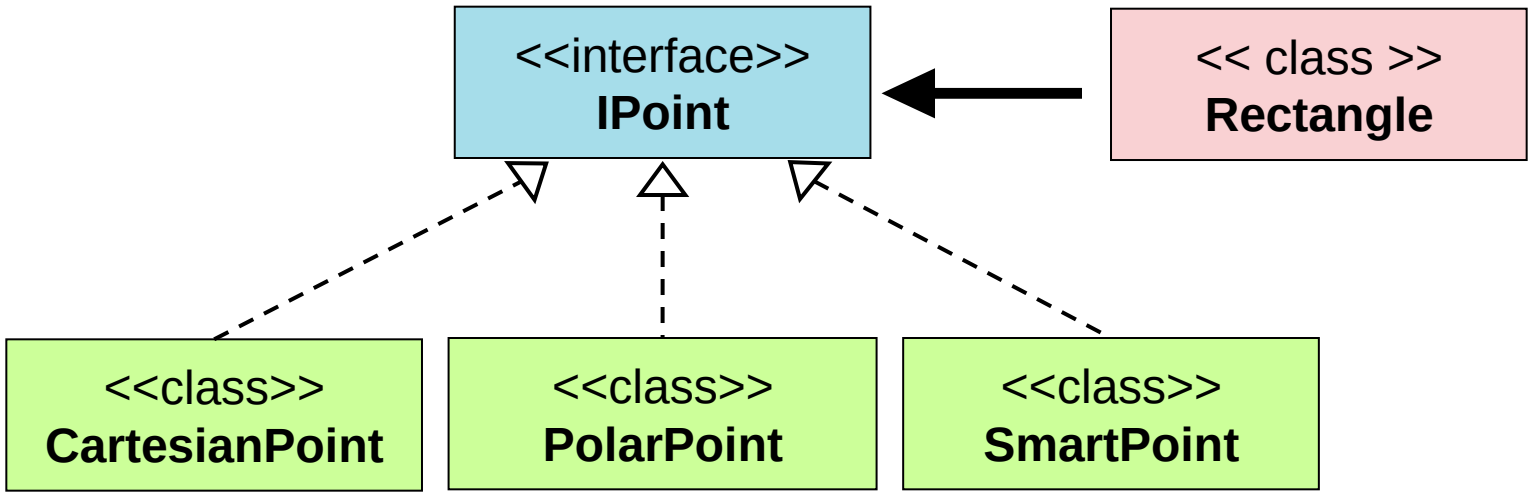


# נראות והורשה

- 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, ואינם מכירים את המחלקות המממשות, יהיו **אדישים** לשינויים עתידים בקוד הספק
- שימוש **במנשקים** חוסך **שכפול בקוד לקוח,** בכך שאותו קטע קוד עובד בצורה נכונה עם מגוון ספקים (פולימורפיזם)

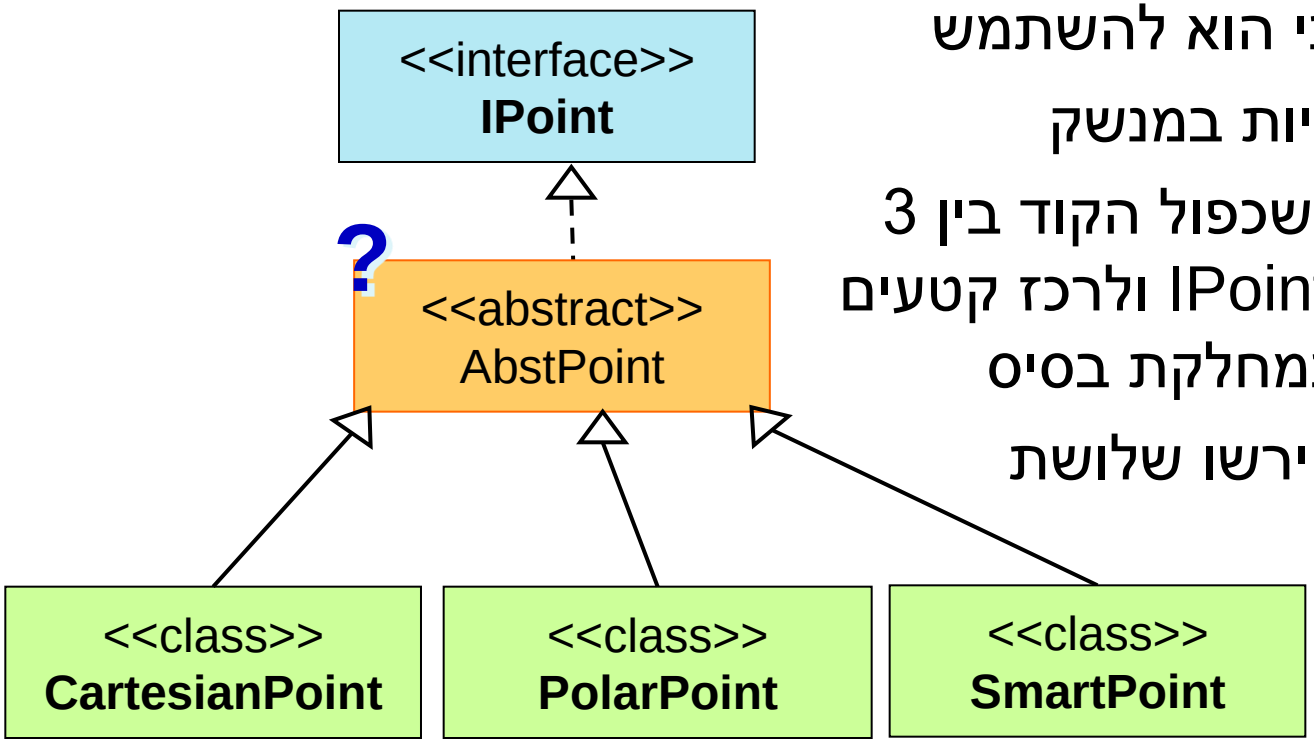


# הממשק IPoint

```
public interface IPoint {

    /** returns the x coordinate of the current point*/
    public double getX();

    /** returns the y coordinate of the current point*/
    public double getY();

    /** returns the distance between the current point and (0,0) */
    public double rho();

    /** returns the angle between the current point and the abscissa */
    public double theta();

    /** move the current point by dx and dy */
    public void translate(double dx, double dy);

    /** rotate the current point by angle degrees with respect to (0,0) */
    public void rotate(double angle);

    …
}
```

# 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
- פתרון אלטרנטיבי הוא להשתמש במתודות דיפולטיות במנשק
- ננסה לזהות את שכפול הקוד בין 3 מימושי המנשק IPoint ולרכז קטעים משותפים אלה במחלקת בסיס משותפת ממנה ירשו שלושת המימושים.



# Abstract Classes מחלקות מופשטות

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

# מחלקות מופשטות - דוגמא

```
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();
```

X

## CartesianPoint

```
private double x;
private double y;
```

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

```
public double getX()  { return x;}
```

```
public double getY()  { 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 getX() {  return r * Math.cos(theta); }
```

```
public double getY() {  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

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

## PolarPoint

```
public double distance(IPoint other) {
    double deltaX = getX()-other.getX();
    double deltaY = getY()-other.getY();

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

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

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

## CartesianPoint

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

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

## PolarPoint

```
public double distance(IPoint other) {
    double deltaX = getX()-other.getX();
    double deltaY = getY()-other.getY();

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

נשאר הבדל אחד:
נחליף את x להיות getX() –
במאזן ביצועים לעומת כלליות נעדיף תמיד את הכלליות

## CartesianPoint

```
public double distance(IPoint other) {
    double deltaX = getX()-other.getX();
    double deltaY = getY()-other.getY();

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

## PolarPoint

```
public double distance(IPoint other) {
    double deltaX = getX()-other.getX();
    double deltaY = getY()-other.getY();

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

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

## CartesianPoint

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

## PolarPoint

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

תהליך דומה ניתן גם לבצע עבור toString

# מימוש המחלקה האבסטרקטית

```
public abstract class AbstractPoint implements IPoint{
    public double distance(IPoint other) {
        double deltaX = getX()-other.getX();
        double deltaY = getY()-other.getY();

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

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

# ירושה מהמחלקה האבסטרקטית

```
public class PolarPoint extends AbstractPoint{
    private double r;
    private double theta;

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

    @Override
    public double getX() {
        return r * Math.cos(theta);
    }

    @Override
    public void rotate(double angle) {
        theta += angle;
    }
     …
}
```