

בית הספר למדעי המחשב
אוניברסיטת תל אביב

תוכנה 1

תרגול מספר 9:

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

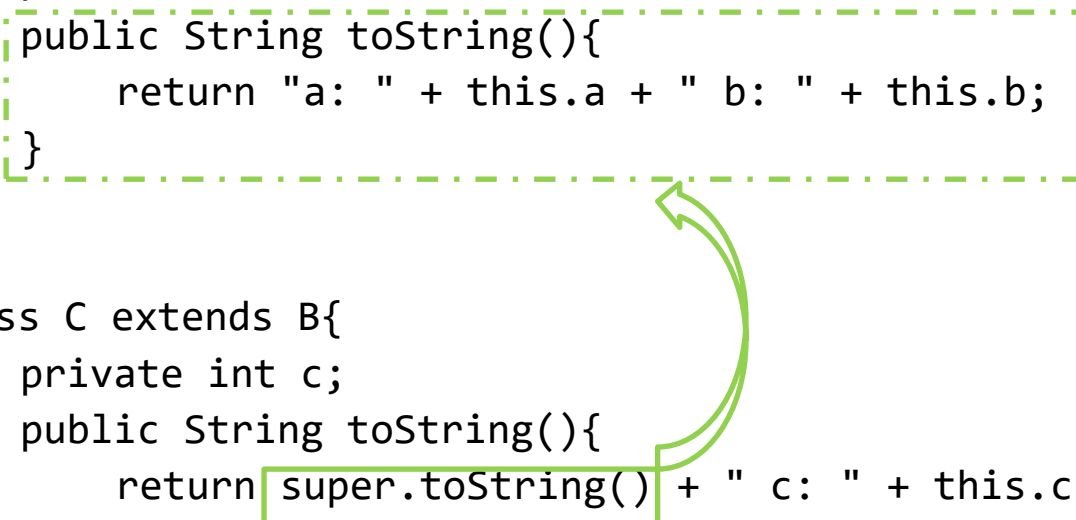
דריסת שירותים

- 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;
    }
}
```

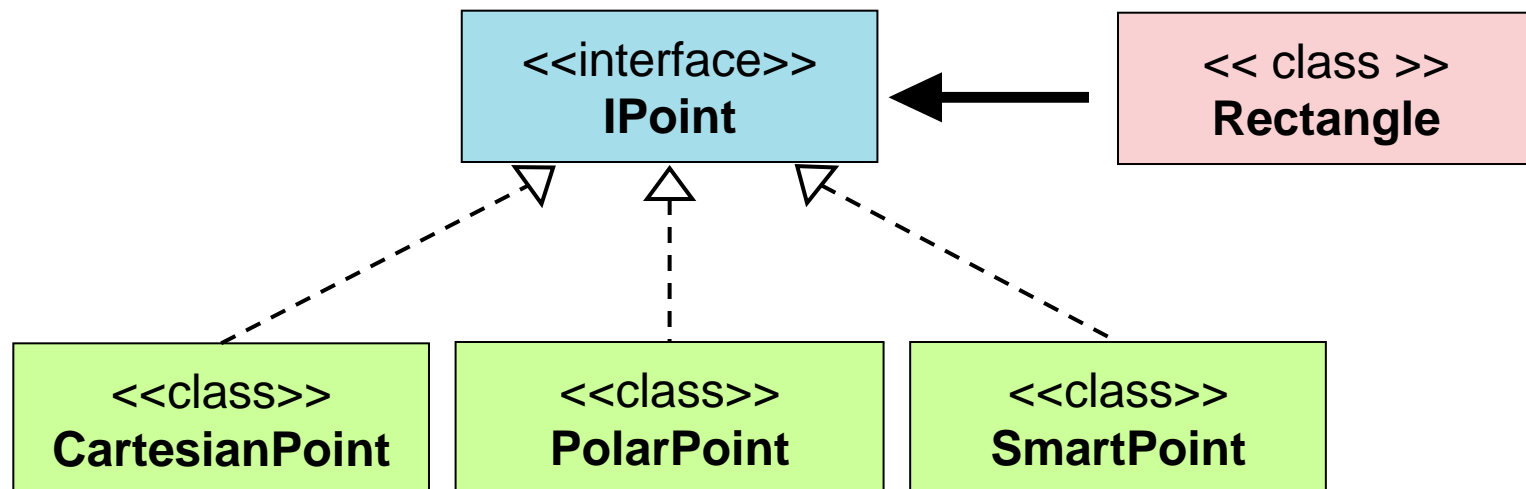


ניראות והורשה

- 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 **אדישיים** לשינויים עתידיים בקוד הספק
- שימוש **במנשקים** חוסך **שכפול בקוד לקוח**, בכך שאותו קטע קוד עובד בצורה נכונה עם מגוון ספקים (פולימורפיזם)

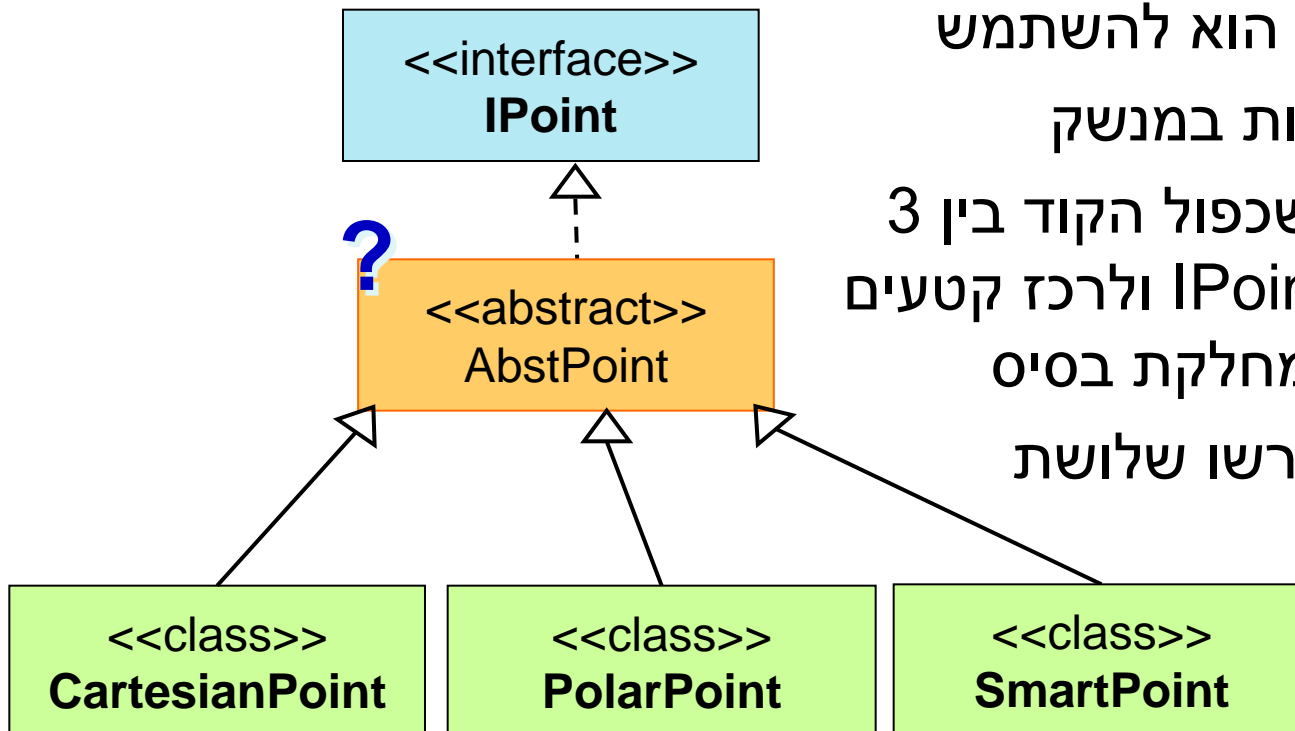


הממשק IPoint

```
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();
```



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...

נוסה לשכתב את CartesianPoint ע"י הוספת משתני העזר ΔX ו- ΔY

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:
 – `getX()` את `x` להיות
 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;
    }
    ...
}
```

חריגים

- נממש שירות המחשב ממוצע הרמוני על אוסף של מספרים.

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \dots + \frac{1}{x_n}}$$

```
public static double harmonicMean(Collection<Integer> numbers){
    if (numbers.isEmpty()){
        return 0;
    }
    double denominator = 0;
    for (int i : numbers){
        denominator += 1.0/i;
    }
    return numbers.size()/ denominator;
}
```

שאלה: ממוצע הרמוני מוגדר רק על מספרים חיוביים. מה נעשה אם נקבל מספר אי-חיובי ברשימה?

חריגים

- אופציה ראשונה:

- נקבל החלטה בתוך השירות, למשל:
- נתעלם מהמספרים האי-חיוביים ונחשב ממוצע הרמוני על שאר המספרים.
- נחזיר 0 או מספר ברירת מחדל אחר
- חסרונות – המשתמש לא ידע שמשהו לא תקין, אם היה יודע, אולי היה מעדיף דרך אחרת לטיפול.

- אופציה שניה:

- שימוש בחריגים.

חריגים

```
public static double harmonicMean(Collection<Integer> numbers) throws Exception{
    if (numbers.isEmpty()){
        return 0;
    }
    double denominator = 0;
    for (int i : numbers){
        if (i <= 0){
            throw new Exception("wrong value in list: " + i);
        }
        denominator += 1.0/i;
    }
    return numbers.size()/denominator;
}
```

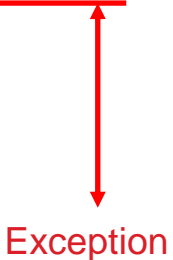
עלינו לייצר אובייקט חדש מטיפוס
Exception ולהשתמש במילה
השמורה throw בשביל לזרוק את
השגיאה

מצהירים על שגיאה
שנזרקת בשירות

חריגים

- נוסף שירות נוסף – השירות מקבל מפה: משם קובץ לאוסף המספרים שהוא מכיל, ומדפיס ממוצע הרמוני עבור כל קובץ.

```
public static void printMeansByFiles(Map<String, Collection<Integer>> numbers) {
    for (Map.Entry<String, Collection<Integer>> mapEntry: numbers.entrySet()){
        double hMeanForFile = harmonicMean(mapEntry.getValue());
        System.out.println("for file: " + mapEntry.getKey() + "
                            hMean is: " + hMeanForFile);
    }
}
```



בקוד הזה יש שגיאת
קומפילציה בגלל שגיאה
שלא הצהרנו עליה אך גם לא
טיפלנו בה

חריגים

- אפשרות ראשונה: לא נטפל בחריג, ורק נצהיר עליו
- במקרה הזה, מי שיצטרך להתמודד עם הטיפול בחריג הוא השירות שיקרא ל `.printMeansByFiles`.

```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo)
    throws Exception{
    for (Map.Entry<String, Collection<Integer>> mapEntry: filesInfo.entrySet()){
        double hMeanForFile = harmonicMean(mapEntry.getValue());
        System.out.println("for file: " + mapEntry.getKey() + "
            hMean is: " + hMeanForFile);
    }
}
```

חריגים

- אפשרות שניה: נטפל בחריג!

```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo) {
    for (Map.Entry<String, Collection<Integer>> mapEntry: filesInfo.entrySet()){
        try{
            double hMeanForFile = harmonicMean(mapEntry.getValue());
            System.out.println("for file: " + mapEntry.getKey() + " hMean is: "
                + hMeanForFile);
        }
        catch (Exception e){
            System.out.println("cannot calculate hMean for file " + mapEntry.getKey());
        }
    }
}
```

חריגים

- איך זה עובד?

```
public static void main(String[] args){  
    Map<String, Collection<Integer>> files = new LinkedHashMap<>();  
    files.put("file1", Arrays.asList(1, 2, 3));  
    files.put("file2", Arrays.asList(1,2,-4));  
    files.put("file3", Arrays.asList(15,17,30));  
    printMeansByFiles(files);  
}
```

- תוכנית זו מייצרת את הפלט:

```
for file: file1 hMean is: 1.6363636363636365  
cannot calculate hMean for file file2  
for file: file3 hMean is: 18.888888888888889
```


חריגים

- ובכל זאת יש בעיה – אנחנו מטפלים בכל שגיאה אפשרית שיכולה להיזרק מתוך `harmonicMean`, ועל הדרך יכולים להתעלם משגיאות שמעידות על באג אפשרי.
- במימוש שלנו הנחנו הנחה סמויה לגבי המפה, למרות שאין לנו דרך לדעת כיצד היא נוצרה (נניח שאין חוזה לשירות).
- מה יקרה במקרה הבא?

```
public static void main(String[] args){
    Map<String, Collection<Integer>> files = new LinkedHashMap<>();
    files.put("file1", null);
    files.put("file2", Arrays.asList(1,2,-4));
    files.put("file3", Arrays.asList(15,17,30));
    printMeansByFiles(files);
}
```

חריגים

```
public static double harmonicMean(Collection<Integer> numbers) throws Exception{  
    if (numbers.isEmpty())  
        ...  
}
```

NullPointerException



```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo)  
    ...  
    catch (Exception e){  
        System.out.println("cannot calculate hMean for file " + mapEntry.getKey());  
    }  
    ...  
}
```

חריגים

- מה נרצה לעשות במידה והמפה שלי מכילה null?
 - יכול להיות שנרצה להתייחס לזה כמו לרשימה ריקה (שזה למעשה הטיפול שקיים כרגע בקוד).
 - יכול להיות שנרצה להדפיס הודעה למשתמש: המפה מכילה null, אולי קרתה שגיאה בטעינת הקובץ?
 - יכול להיות שנרצה לזרוק את השגיאה ולהטיל את הטיפול על מי שמשתמש ב `printMeansByFiles`
- אם נרצה להתייחס למקרה של מפה המכילה null באופן שונה ממפה המכילה מספר לא חיובי, עלינו לדעת להבדיל בין החריגים.
 - הצעה: נוסיף בלוק `except` עבור `NullPointerException`
 - ומה אם יש עוד שגיאות שיכולות להיזרק?

יצירת טיפוס חריג חדש

ירושה מ Exception

```
class HMeanException extends Exception{  
    public HMeanException(String message) {  
        super("Harmonic Mean calculation error! " + message);  
    }  
}
```

קריאה לבנאי של מחלקת האב – קריאה זו תמיד
תהיה הפקודה הראשונה של הבנאי

שימוש בטיפוס החרוג החדש

```
public static double harmonicMean(Collection<Integer> numbers) throws HMeanException {  
    if (numbers.isEmpty()){  
        return 0;  
    }  
    double denominator = 0;  
    for (int i : numbers){  
        if (i <= 0){  
            throw new HMeanException("wrong value in list: " + i);  
        }  
        denominator+ = 1.0/i;  
    }  
    return numbers.size()/denominator;  
}
```

שימוש בטיפוס החרוג החדש

```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo)
...
catch (HMeanException e){
    System.out.println("cannot calculate hMean for file " + mapEntry.getKey());
}
...
}
```

הבלוק הזה יטפל רק
בשגיאה שזרקנו מתוך
harmonicMean, חריגים
אחרים יזרקו הלאה.

שימוש בשגיאות – פרמט הודעת השגיאה

```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo)
    ...
    catch (HMeanException e){
        System.out.println("cannot calculate hMean for file " + mapEntry.getKey());
        e.printStackTrace();
    }
}
public static void main(String[] args){
    Map<String, Collection<Integer>> files = new LinkedHashMap<>();
    files.put("file2", Arrays.asList(1,2,-4));
    files.put("file3", Arrays.asList(15,17,30));
    printMeansByFiles(files);
}
```

```
for file: file1 hMean is: 1.6363636363636365
cannot calculate hMean for file file2
HMeanException: Harmonic Mean calculation error! wrong value in list: -4
    at Tmp.harmonicMean(Tmp.java:22)
    at Tmp.printMeansByFiles(Tmp.java:36)
    at Tmp.main(Tmp.java:52)
for file: file3 hMean is: 18.888888888888889
```

• עבור תוכנית זו נקבל את הפלט:

שימוש בשגיאות

- הדפסת פורמט שגיאה מצומצם יותר:

```
public static void printMeansByFiles(Map<String, Collection<Integer>> fileInfo)
...
    catch (HMeanException e){
        System.out.println("cannot calculate hMean for file " + mapEntry.getKey());
        System.out.println(e.getMessage())
    }
}
```

- פלט התוכנית יהיה:

```
cannot calculate hMean for file file2
Harmonic Mean calculation error! wrong value in list: -4
for file: file3 hMean is: 18.88888888888889
```

```
class HMeanException extends Exception{
    public HMeanException(String message) {
        super("Harmonic Mean calculation error! " + message);
    }
}
```