



תוכנה 1 בשפת Java

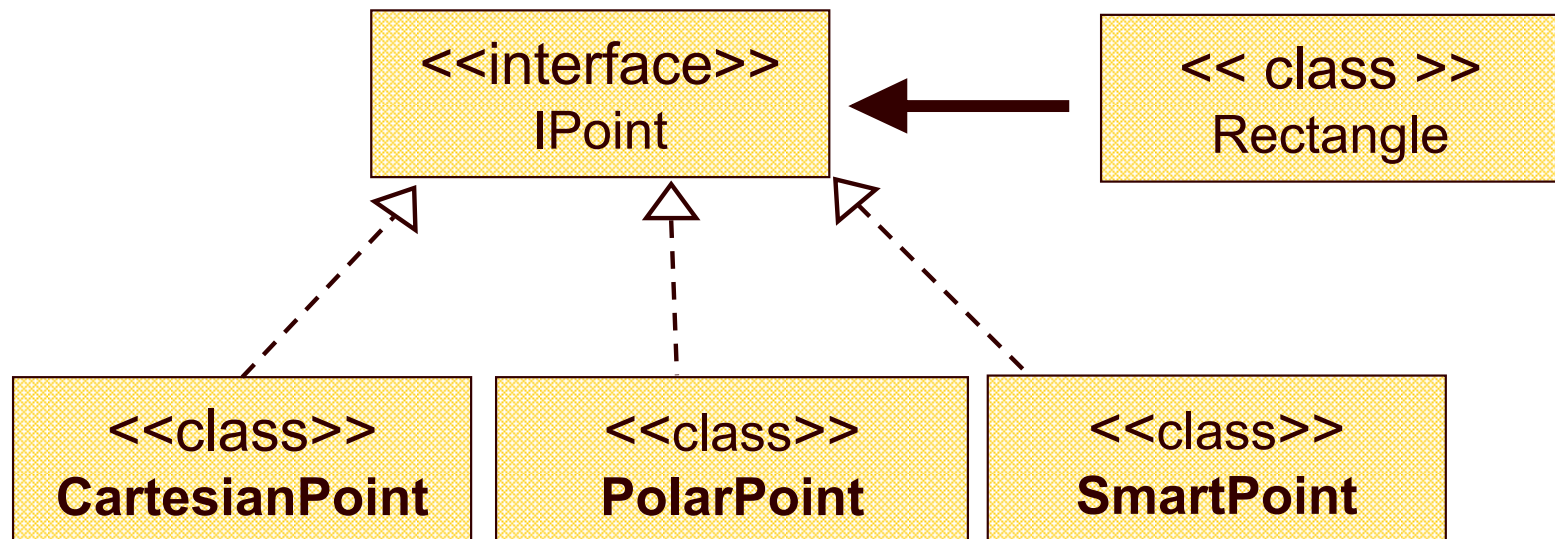
תרגול מספר 8: תבניות הורשה ומימוש

ליאור שפירא
אוהד ברזילי

בית הספר למדעי המחשב
אוניברסיטת תל אביב

צד הלקוח

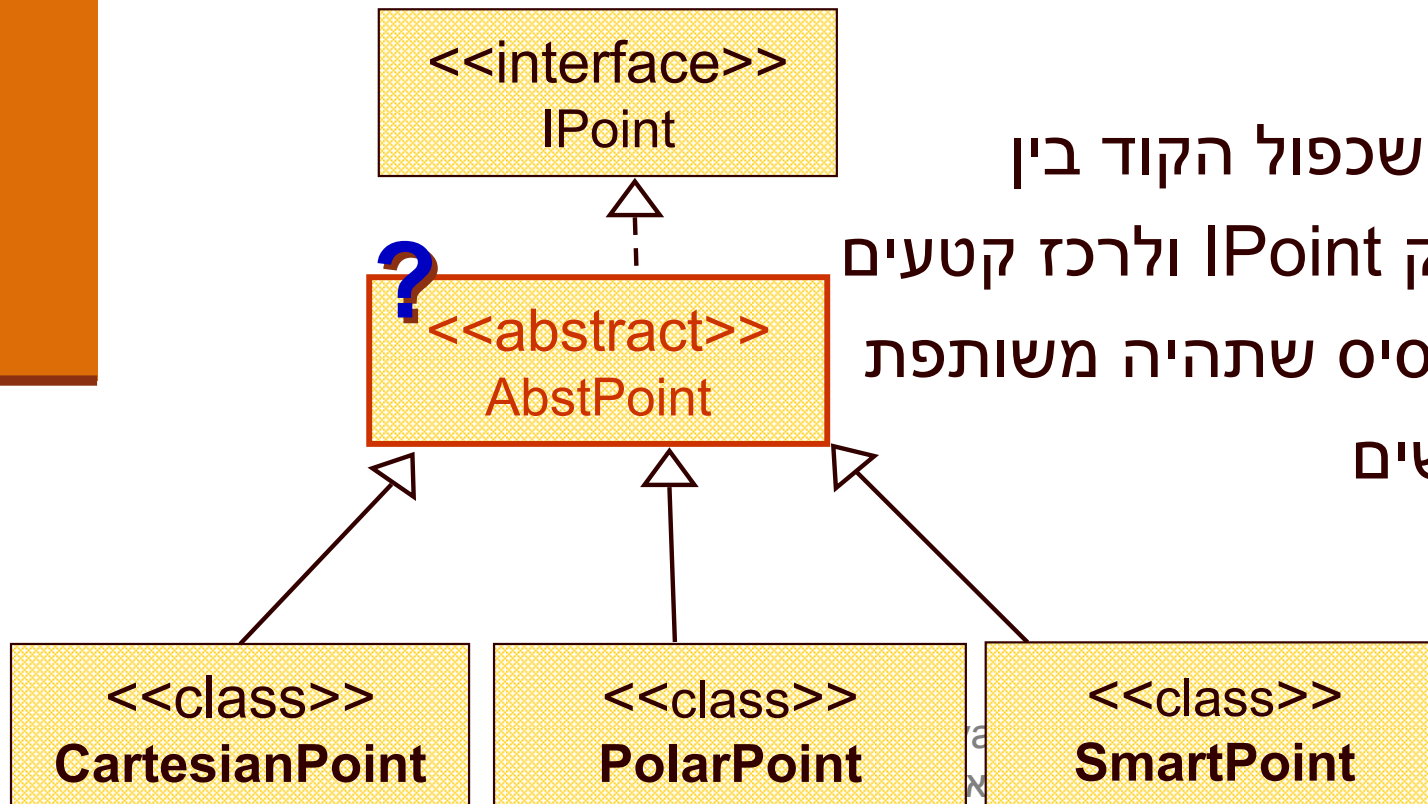
- בהרצאה ראינו את המנשק IPoint, והצגנו 3 מימושים שונים עבורו
- ראינו כי לקוחות התלויים במנשק IPoint בלבד, ולא מכירים את המחלקות המממשות אדישיים לשינויים עתידיים בקוד הספק
- שימוש במנשקים חוסך שכפול קוד לקוח, בכך שאותו קטע קוד עובד בצורה נכונה עם מגוון ספקים (פולימורפיזם)



צד הספק

■ בהרצאה האחרונה היכרנו את **מנגנון ההורשה** אשר 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

```
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; }
```

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);  
}
```

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);  
}
```


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) {  
    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);  
}
```

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);  
}
```

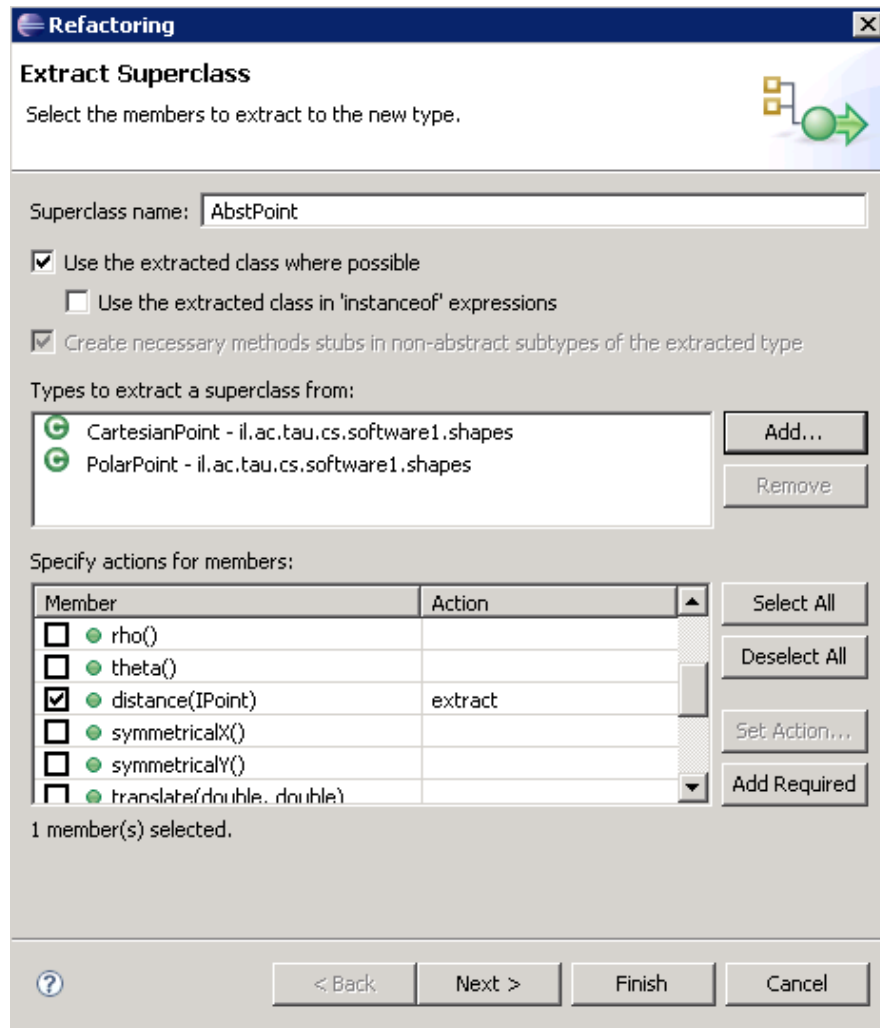
CartesianPoint

```
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 + ")";  
}
```

Extract Superclass Refactoring



ניתן לבצע תהליך זה בצורה
אוטומטית ע"י שכתוב מבני
(Refactoring) שנקרא:
Extract Superclass

הגרסה ב- Eclipse עוד לא
"מושלמת"



Adapter Design Pattern

מחשבים בחו"ל



■ יצאת לחו"ל (ברצלונה) עם המחשב הנייד

■ זכרת להביא כבל חשמל

■ תואם בהספק (220v)

■ תואם בתדר (50hz)



■ אבל כל זה לא מספיק,

■ אם לא הבאת מתאם





בעיית המתאם

הבעיה היא טכנית

- הלקוח (laptop) זקוק ל 220v ב- 50hz
- הספק (רשת החשמל) מספק 220v ב- 50hz
- הבעיה היא בממשק בין השניים

בעיה דומה עשויה לקרות גם בתוכנה

- במערכת חדשה אנו זקוקים ליכולות מסוימות (למחלקה שממשת משהו)
- קיים מודול (מחלקה) שנכתב למערכת קודמת המספק את היכולות חשובות
- אולם בשלב התכנון של המערכת החדשה הוגדר ממשק למחלקה זו השונה מהממשק שאותו מממש המודול הקיים


```
public interface RandomNumberGenerator {  
    public int random();  
}
```

דוגמא

אפשר היה לממש את השירות בעצמנו, למשל כך: ■

```
public class FastRandom implements RandomNumberGenerator {  
  
    public int last;  
  
    public FastRandom() {  
        last = (int) System.nanoTime();  
    }  
  
    public int random() {  
        last = last * 1103515245 + 12345;  
        return (short) (last & 0xFFFF);  
    }  
  
}
```

שימוש חוזר

■ אולם הרבה יותר פשוט להשתמש במחלקה קיימת:

■ `java.util.Random`

■ בעיה:

■ המחלקה אינה מממשת את המנשק הדרוש במערכת:

`RandomNumberGenerator`

■ ובפרט היא איננה מממשת את השרות הדרוש `random()`

Method Summary	
<code>protected int</code>	<code>next(int bits)</code> Generates the next pseudorandom number.
<code>boolean</code>	<code>nextBoolean()</code> Returns the next pseudorandom, uniformly distributed <code>boolean</code> value from this random number generator's sequence.
<code>void</code>	<code>nextBytes(byte[] bytes)</code> Generates random bytes and places them into a user-supplied byte array.
<code>double</code>	<code>nextDouble()</code> Returns the next pseudorandom, uniformly distributed <code>double</code> value between 0.0 and 1.0 from this random number generator's sequence.
<code>float</code>	<code>nextFloat()</code> Returns the next pseudorandom, uniformly distributed <code>float</code> value between 0.0 and 1.0 from this random number generator's sequence.
<code>double</code>	<code>nextGaussian()</code> Returns the next pseudorandom, Gaussian ("normally") distributed <code>double</code> value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.
<code>int</code>	<code>nextInt()</code> Returns the next pseudorandom, uniformly distributed <code>int</code> value from this random number generator's sequence.
<code>int</code>	<code>nextInt(int n)</code> Returns a pseudorandom, uniformly distributed <code>int</code> value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.
<code>long</code>	<code>nextLong()</code> Returns the next pseudorandom, uniformly distributed <code>long</code> value from this random number generator's sequence.
<code>void</code>	<code>setSeed(long seed)</code> Sets the seed of this random number generator using a single <code>long</code> seed.



המתאם

- הבעיה היא טכנית והיא נעוצה בממשק:
 - המחלקה המבוקשת נדרשת לממש את `random()`
 - ואילו `java.util.Random`, אף על פי שהוא מממש אקראיות, הוא עושה זאת בעזרת פונקציות `nextXXX()`
- הפתרון: מחלקת מתאם (Adapter)

```
public class RandomAdapter implements RandomNumberGenerator {  
  
    java.util.Random r;  
  
    public RandomAdapter() {  
        r = new java.util.Random();  
    }  
  
    public int random() {  
        return r.nextInt();  
    }  
}
```

המחלקה מתאמת (מתרגמת) בין
המנשק של המחלקה `Random` למנשק
`RandomNumberGenerator`



לא רק תאומים

■ ומה אם נרצה לצאת עם המחשב הנייד לארה"ב?

■ כעת הבעיה היא לא רק בממשק

■ אלא גם בהפרשי המתחים

■ ניתן לפתור בעיה זו באותה שיטה:

■ המתאם (adapter) יבצע גם את המרת הזרם (transformator)
ע"י שימוש בספק המקורי

■ בדוגמא שלנו: איך נממש מנשק חדש שבו נדרשים להחזיר זוג
(Set) מספרים אקראיים?

```
import java.util.Set;

public interface RandomSetGenerator {
    public Set<Integer> random();
}
```

```
import java.util.Set;
import java.util.TreeSet;

public class RandomSetAdapter implements RandomSetGenerator {

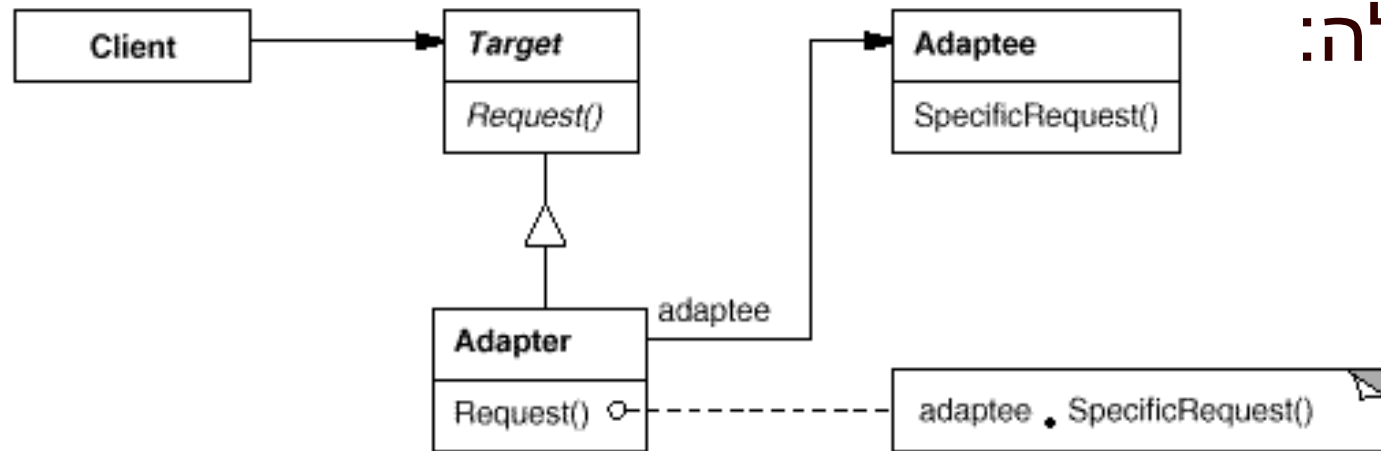
    java.util.Random r;

    public RandomSetAdapter() {
        r = new java.util.Random();
    }

    public Set<Integer> random() {
        Set<Integer> result = new TreeSet<Integer>();
        result.add(r.nextInt());
        result.add(r.nextInt());
        return result;
    }
}
```

סוגים של מתאמים

■ מבוססי הכלה:



■ מבוססי הורשה:

