מתאמים ומחלקות פנימיות

תרגול 10

ליאור שפירא, מתי שמרת

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 <u>למערכת קודמת</u> המספק את היכולות חשובות
  - אולם בשלב התכנון של המערכת החדשה הוגדר מנשק למחלקה זו השונה מהמנשק שאותו ממש 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()

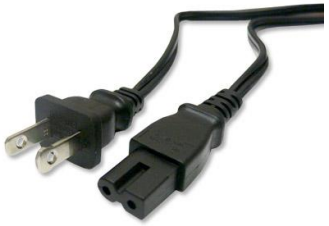
Method Summary	
protected int	next(int bits) Generates the next pseudorandom number.
boolean	nextBoolean() Returns the next pseudorandom, uniformly distributed boolean value from this random number generator's sequence.
void	nextBytes(byte[] bytes) Generates random bytes and places them into a user-supplied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

המתאם

- הבעיה היא טכנית והיא נעוצה בממשק:
  - המחלקה המבוקשת נדרשת למממש 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

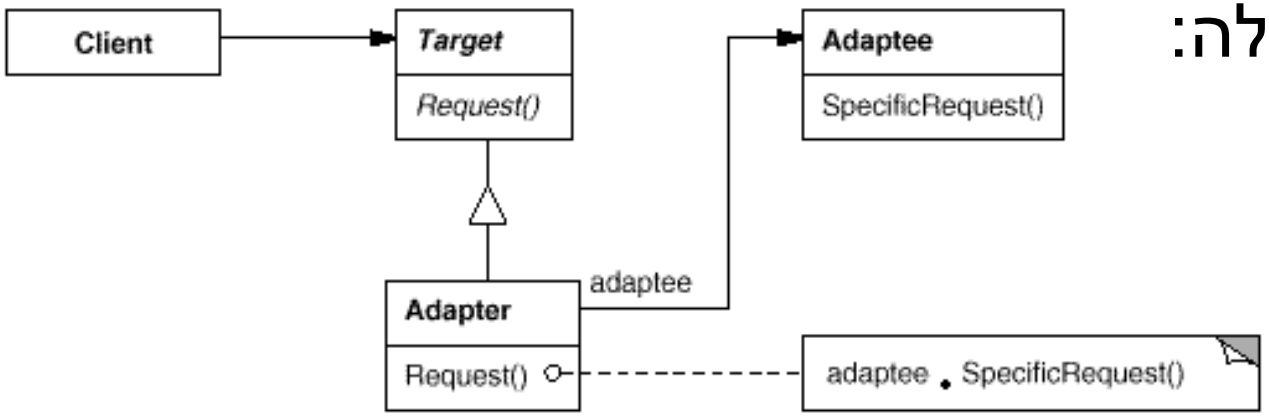
- ומה אם נרצה לצאת עם המחשב הנייד לארה"ב?
- כעת הבעיה היא לא רק בממשק
  - אלא גם בהפרשי המתחים
- ניתן לפתור בעיה זו באותה שיטה:
  - המתאם (adapter) יבצע גם את המרת הזרם (tranformator) ע"י שימוש בספק המקורי
- בדוגמא שלנו: איך נממש מנשק חדש שבו נדרשים להחזיר **זוג** (Set) מספרים אקראיים?

```java
import java.util.Set;

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

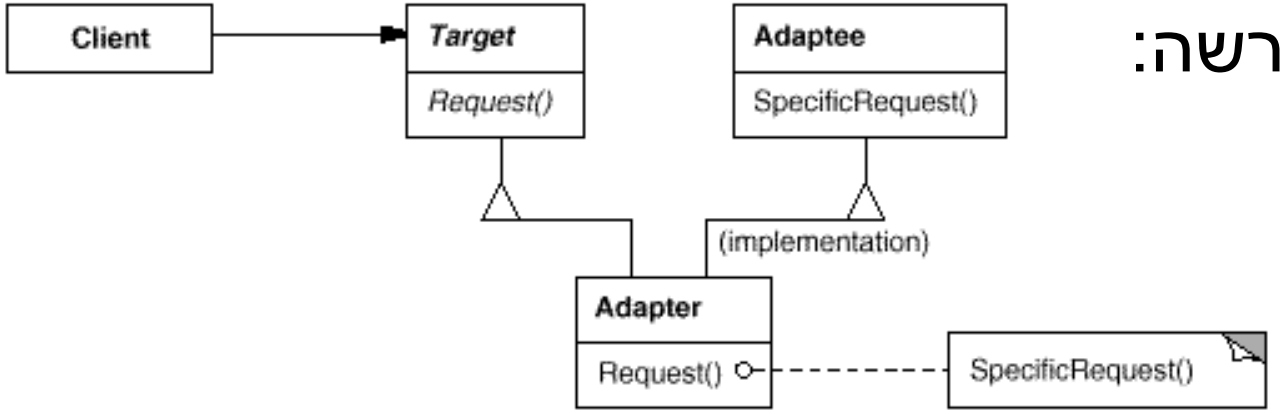
```java
import java.util.Set;
import java.util.TreeSet;

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:



מחלקות פנימיות (מקוננות)
Inner (Nested) Classes

Inner Classes

- מחלקה פנימית היא מחלקה שהוגדרה בתחום (Scope – בין המסולסליים) של מחלקה אחרת

- דוגמא:

```
public class House {
    private String address;
    public class Room {
        private double width;
        private double height;
    }
}
```

שימוש לב!

Room אינה שדה של המחלקה House

מחלקות פנימיות

- הגדרת מחלקה כפנימית מרמזת על היחס בין המחלקה הפנימית והמחלקה העוטפת:
  - למחלקה הפנימית יש משמעות רק בהקשר של המחלקה החיצונית
  - למחלקה הפנימית יש הכרות אינטימית עם המחלקה החיצונית
  - המחלקה הפנימית היא מחלקת עזר של המחלקה החיצונית

- דוגמאות:
  - `Collection` ו- `Iterator`
  - `Body` ו- `Brain`
  - מבני נתונים המוגדרים ברקורסיה: `List` ו- `Cell`

Inner Classes

- **ב Java כל מופע של עצם מטיפוס המחלקה הפנימית צריך להיות משויך לעצם מטיפוס המחלקה העוטפת**

- **השלכות**
  - תחביר מיוחד לבנאי
  - לעצם מטיפוס המחלקה הפנימית יש שדה הפנייה שמיוצר אוטומטית לעצם מהמחלקה העוטפת
  - כתוצאה לכך יש למחלה הפנימית גישה לשדות ולשרותים (אפילו פרטיים!) של המחלקה העוטפת **ולהיפך**

Inner Classes

```
public class House {
    private String address;
    public class Room {
            // hidden reference to a House
            private double width;
            private double height;
            public String toString(){
                    return  "Room inside: " +  address;
            }
    }
}
```

Inner Classes

```
public class House {
    private String address;
    private double  height;
    public class Room {
            // hidden reference to a House
            private double height;
            public String toString(){
                    return  "Room height: " + height
                    + " House height: " + House.this.height;
            }
    }
}
```

Height of *House*

Height of *Room*

Height of *Room*
Same as *this.height*

יצירת מופעים

- כאשר המחלקה החיצונית יוצרת מופע של עצם מטיפוס המחלקה הפנימית אזי העצם יווצר בהקשר של העצם היוצר

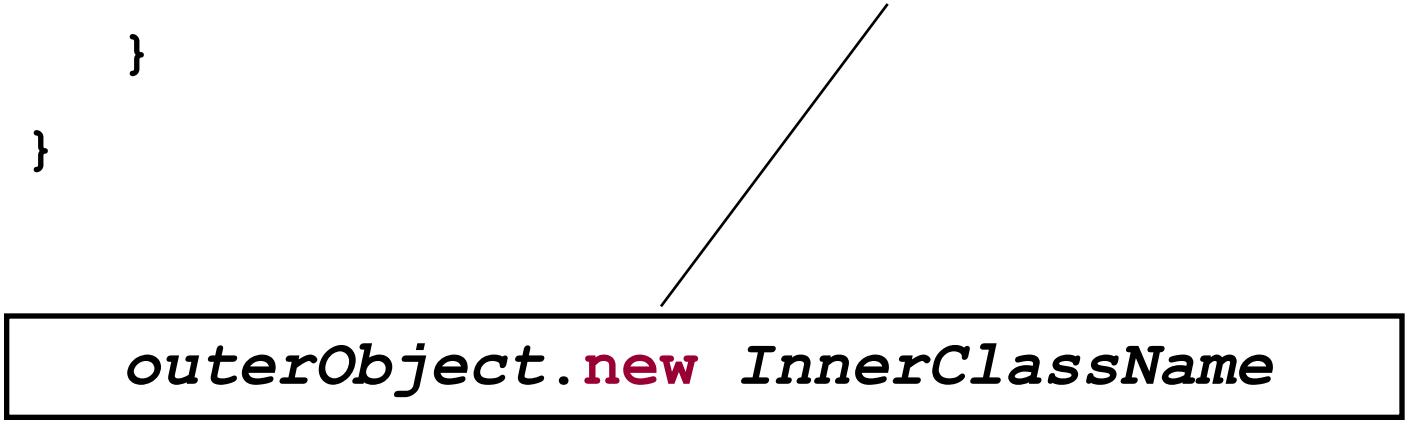
- כאשר עצם מטיפוס המחלקה הפנימית נוצר מחוץ למחלקה העוטפת, יש צורך בתחביר מיוחד

יצירת מופע ע"י המחלקה החיצונית

```
public class House {
    private String address;
    public void test() {
            Room  r = new Room();
            System.out.println( r );
    }

    public class Room {
            ...
    }
}
```

יצירת מופע שלא ע"י המחלקה החיצונית

```
public class Test {
    public static void main(String[] args){
        House  h = new House();
        House.Room  r =  h.new Room();
    }
}
```



Static Nested Classes

- ניתן להגדיר מחלקה פנימית כ **`static`** ובכך לציין שהיא אינה קשורה למופע מסוים של המחלקה העוטפת

- הדבר אנלוגי למחלקה שכל שרותיה הוגדרו כ **`static`** והיא משמשת **כספריה** עבור מחלקה מסוימת

- בשפת C++ יחס זה מושג ע"י הגדרת יחס **`friend`**

```
public class House {
    private String address;
    public static class Room {
        public String toString(){
            return  "Room " +  address;
        }
    }
}
```

Error: this room is not related to any house

```
public class Test {
    public static void main(String[] args){
        House.Room  r =  new House.Room();
        ...
```

Not related to any house

new *OuterClassName.InnerClassName*

הגנה על מחלקות פנימיות סטאטיות

- אם המחלקה הפנימית אינה ציבורית (אינה מוגדרת public), הטיפוס שלה מוסתר, אבל עצמים מהמחלקה אינם מוסתרים אם יש התייחסות אליהם

```
public class Outer … {
   private static class Inner implement Inter {…}
   public static Inter getInner() {
      return new Inner ();
   }
    …

Inter i = new Outer.Inner();   //error
Inter i = Outer.getInner();    // ok
```

מחלקות מקומיות - מחלוקת פנימיות בתוך מתודות

- ניתן להגדיר מחלקה פנימית בתוך מתודה של המחלקה החיצונית
- הדבר מגביל את תחום ההכרה של אותה מחלקה לתחום אותה המתודה בלבד
- המחלקה הפנימית תוכל להשתמש במשתנים מקומיים של המתודה רק אם הם הוגדרו כ **`final`** (מדוע?)

מחלקות מקומיות

```
public class Test {
    ...
   public void test () {
           class Info {
                  private int x;
                  public Info(int x) {this.x=x;}
                  public String toString() {
                         return "** " + x + "**" ;
                  }
           };
           Info  inf1 = new Info(0);
           System.out.println(inf1);
    }
}
```

שימוש במשתנים מקומיים

```
public class Test {
    public void test (int x) {
        final int y = x+3;
        class Info {
            public String toString(){
                    return "**" +  y + "***";
            }
        };
        System.out.println( new Info());
    }
}
```

מחלקות אנונימיות

- בעזרת מחלקות פנימיות ניתן להגדיר מחלקות אנונימיות – מחלקות ללא שם

- מחלקות אנונימיות שימושיות מאוד במערכות מונחות ארועים (כגון GUI) וילמדו בהמשך הקורס

הידור של מחלקות פנימיות

- המהדר (קומפיילר) יוצר קובץ class. עבור כל מחלקה. מחלקה פנימית אינה שונה במובן זה ממחלקה רגילה
- שם המחלקה הפנימית יהיה **Outer$Inner.class**
- אם המחלקה הפנימית אנונימית, שם המחלקה שיוצר הקומפיילר יהיה **Outer$1.class**