

תוכנה 1

תרגול מספר 11:

Static vs. Dynamic Binding

מחלקות מקוננות Nested Classes





STATIC VS. DYNAMIC BINDING

Static versus Dynamic Binding

- ```
public class Account {
 public String getName(){...};
 public void deposit(int amount) {...};
}
```

```
public class SavingsAccount extends Account {
 public void deposit(int amount) {...};
}
```
- ```
Account obj = new Account();  
obj.getName();  
obj.deposit(...);
```



```
Account obj = new SavingsAccount();  
obj.getName();  
obj.deposit(...);
```

Which version is called ?

Binding in Java

- Binding is the process by which references are bound to specific classes.
- Used to resolve which methods and variables are used at **run time**.
- There are two kind of bindings: static binding and dynamic binding.

Binding in Java

- **Static Binding (Early Binding)**

- The compiler can resolve the binding at compile time. (As in the previous example)

- **Dynamic Binding (Late Binding)**

- The compiler is not able to resolve the call and the binding is done at runtime only.
- *Dynamic dispatch*

Static binding (or early binding)

- Static binding: bind at compilation time
- Performed if the compiler can resolve the binding at compile time
- Applied for
 - Static methods
 - Private methods
 - Final methods
 - Fields

Static binding example – Static methods

```
public class A {  
    public static void m() {  
        System.out.println ("A");  
    }  
}  
  
public class B extends A {  
    public static void m() {  
        System.out.println ("B");  
    }  
}  
  
public class StaticBindingTest {  
    public static void main(String args[]) {  
        A.m();  
        B.m();  
  
        A a = new A();  
        A b = new B();  
        a.m();  
        b.m();  
    }  
}
```

Output:

A
B
A
A

Static binding example - Fields

```
public class A {  
    public String someString = "member of A";  
}
```

```
public class B extends A {  
    public String someString = "member of B";  
}
```

```
public class StaticBindingTest {  
    public static void main(String args[]) {  
  
        A a = new A();  
        A b = new B();  
        B c = new B();  
  
        System.out.println( a.someString );  
        System.out.println( b.someString );  
        System.out.println( c.someString );  
    }  
}
```

Output:

```
member of A  
member of A  
member of B
```


Dynamic Binding

- `void func (Account obj) {
 obj.deposit();
}`
- What should the compiler do here?
 - The compiler doesn't know which concrete object type is referenced by `obj`
 - The method to be called can only be known at run time (*because of polymorphism and method overriding*)
 - Run-time binding

Dynamic Binding

```
public class DynamicBindingTest {  
    public static void main(String args[]) {  
        Vehicle vehicle = new Car();           //The reference type is Vehicle but run-time object will be Car  
        vehicle.start();                       //Car's start called because start() is overridden method  
    }  
}  
  
class Vehicle {  
    public void start() {  
        System.out.println("Inside start method of Vehicle");  
    }  
}  
  
class Car extends Vehicle {  
    @Override  
    public void start() {  
        System.out.println("Inside start method of Car");  
    }  
}
```

Output: "Inside start method of Car"

difference between static and dynamic binding

- Static binding happens at compile-time while dynamic binding happens at runtime.
- Binding of private, static and final methods always happen at compile time since these methods cannot be overridden. Binding of overridden methods happen at runtime.
- Java uses static binding for overloaded methods and dynamic binding for overridden methods.

```
class Outer {
    static class NestedButNotInner {
        ...
    }
    class Inner {
        ...
    }
}
```

מחלקות מקוננות

NESTED CLASSES

מחלקה מקוננת (Nested Class)

■ מחלקה מקוננת היא מחלקה המוגדרת בתוך מחלקה אחרת.

■ סוגים:

- מחלקות פנימיות (inner) {
1. סטטית (static member)
 2. לא סטטית (non-static member)
 3. אנונימית (anonymous)
 4. מקומית (local)

בשביל מה זה טוב ?

■ קיבוץ לוגי

אם משתמשים בטיפוס מסוים רק בהקשר של טיפוס אחר, נטמיע את הטיפוס כדי לשמר את הקשר הלוגי.

■ הכמסה מוגברת

על ידי הטמעת טיפוס אחד באחר אנו חושפים את המידע הפרטי רק לטיפוס המוטמע ולא לכולם.

■ קריאות

מיקום הגדרת טיפוס בסמוך למקום השימוש בו.

מחלקות מקוננות - תכונות משותפות

- למחלקה מקוננת יש גישה לשדות הפרטיים של המחלקה העוטפת ולהיפך
- הנראות של המחלקה היא עבור "צד שלישי"
- אלו הן מחלקות (כמעט) רגילות לכל דבר ועניין
- יכולות להיות אבסטרקטיות, לממש מנשקים, לרשת ממחלקות אחרות וכדומה

Static Member Class

- מחלקה רגילה ש"במקרה" מוגדרת בתוך מחלקה אחרת
- החוקים החלים על איברים סטטיים אחרים חלים גם על מחלקות סטטיות
 - גישה לשדות / פונקציות סטטיים בלבד
 - גישה לאיברים לא סטטיים רק בעזרת הפניה לאובייקט
- גישה לטיפוס בעזרת שם המחלקה העוטפת

```
OuterClass.StaticNestedClass
```

יצירת אובייקט

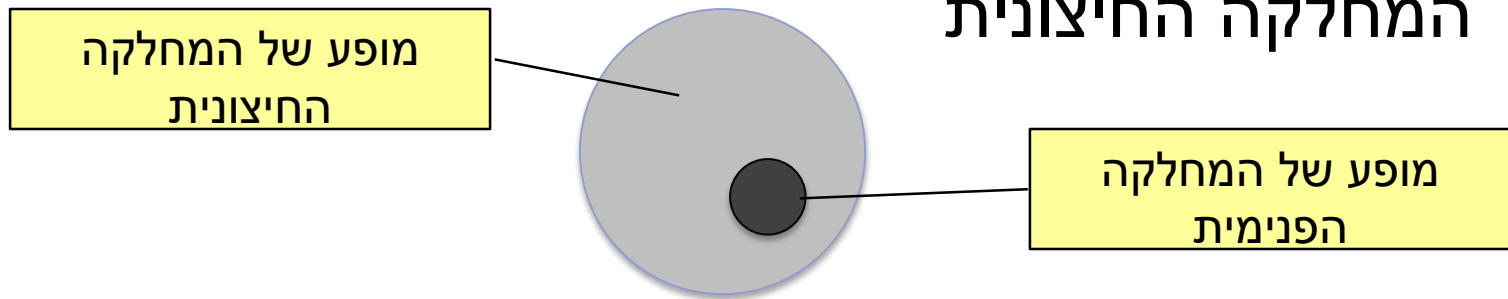
```
OuterClass.StaticNestedClass nested =  
    new OuterClass.StaticNestedClass ();
```


AbstractMap

```
public abstract class AbstractMap<K,V> implements Map<K,V> {  
  
    public static class SimpleEntry<K,V>  
        implements Entry<K,V>, java.io.Serializable {  
        private final K key;  
        private V value;  
  
        ...  
    }  
  
    ...  
}
```

Non-static Member Class

כל מופע של המחלקה הפנימית משויך למופע של המחלקה החיצונית



השיוך מבוצע בזמן יצירת האובייקט ואינו ניתן לשינוי

באובייקט הפנימי קיימת הפניה לאובייקט החיצוני (qualified this)

House Example

```
public class House {
    private String address;

    public class Room {
        // implicit reference to a House
        private double width;
        private double height;

        public String toString(){
            return "Room inside: " + address;
        }
    }
}
```

גישה למשתנה פרטי לא סטטי

Inner Classes

```
public class House {  
    private String address;  
    private double height;  
    public class Room {  
        // implicit 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
Same as `this.height`

AbstractList

```
public abstract class AbstractList<E> extends
    AbstractCollection<E> implements List<E> {
    public Iterator<E> iterator() {
        return new Itr();
    }

    private class Itr implements Iterator<E> {
        ...
    }

    private class ListItr extends Itr implements
        ListIterator<E> {
        ...
    }
}
```

יצירת מופעים

■ כאשר המחלקה העוטפת יוצרת מופע של עצם מטיפוס המחלקה הפנימית אזי העצם נוצר בהקשר של העצם היוצר

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

`outerObject.new InnerClassConstructor(...)`

מחלקות אנונימיות

- מחלקה ללא שם
- הגדרה ויצירת מופע בנקודת השימוש
- מגבלות:
 - חייבת לרשת מטיפוס קיים (מנשק או מחלקה)
 - לא ניתן להגדיר איברים סטטיים, לא ניתן להשתמש בהקשר שדורש שם (instanceof), לא ניתן לממש מספר מנשקים, לקוחות מוגבלים לממשק של טיפוס האב, גישה למשתנים מקומיים שהם final בלבד.
- מחלקה אנונימית צריכה להיות קצרה כדי לא לפגוע בקריאות של הקוד

דוגמאות שימוש

Function object (functor) ■

מיון מחרוזות לפי אורך ■

```
Arrays.sort(stringArray, new Comparator<String>() {  
    public int compare(String s1, String s2) {  
        return s1.length() - s2.length();  
    }  
});
```

```
public Iterator<E> iterator() {  
    return new Iterator<E>() {  
        boolean hasNext() {...}  
        E next() {...}  
        void remove() {...}  
    };  
}
```

מימוש איטרטור ■

המרה ממערך לרשימה

```
static List<Integer> intArrayAsList(final int[] a) {  
    if (a == null)  
        throw new IllegalArgumentException();  
    return new AbstractList<Integer>() {  
        public Integer get(int i) {  
            return a[i];  
        }  
        public Integer set(int i, Integer val) {  
            int oldVal = a[i];  
            a[i] = val;  
            return oldVal;  
        }  
        public int size() {  
            return a.length;  
        }  
    };  
}
```

גישה למשתנים מקומיים
שהוגדרו final
או שהם final באופן אפקטיבי
(לא משנים הצבעה)

מחלקות מקומיות

■ מוגדרות בתוך מתודות

■ יש להם שם וניתן להשתמש בהם מספר פעמים, בתוך אותה מתודה

■ אובייקט עוטף רק אם הוגדרו בהקשר לא סטטי; לא ניתן להגדיר משתנים סטטיים

■ המחלקה הפנימית תוכל להשתמש גם במשתנים מקומיים של המתודה אבל רק אם הם הוגדרו כ-
final

דוגמא - המרה ממערך לרשימה

```
static List<Integer> intArrayAsList(final int[] a) {
    if (a == null)
        throw new IllegalArgumentException();
    class IntegerList extends AbstractList<Integer> {
        public Integer get(int i) {
            return a[i];
        }
        public Integer set(int i, Integer val) {
            int oldVal = a[i];
            a[i] = val;
            return oldVal;
        }
        public int size() {
            return a.length;
        }
    }
    return new IntegerList();
}
```

הסוף...