Design Patterns

- A general reusable solution to recurring design problems.
- Not a recipe
- A higher level language for design
- Factory, Singleton, observer and not “this class inherits from that other class”
- *Design Patterns: Elements of Reusable Object-Oriented Software*
- Lots of information online

Different Views

- When the data change all views should change
- Views dependant on data
- Views may vary, more added in the future
- Data store implementation may changes
- We want:
  - Separate the data aspect from the view one
  - Notify views upon change in data

The Observer Design Pattern

- A.k.a publish/subscribe

Observer and Java

- Java provides an **observer** interface and an **Observable** class.
- Subclass Observable to implement your own subject
- registration and removal of observers
- notification
- Implement Observer
- Other uses of this pattern throughout the JDK
Example Code - Subject

```java
public class IntegerDataBag extends Observable implements Iterable<Integer> {
    private ArrayList<Integer> list = new ArrayList<Integer>();
    public void add(Integer i) {
        list.add(i);
        setChanged();
        notifyObservers();
    }
    public Iterator<Integer> iterator() {
        return list.iterator();
    }
    public Integer remove(int index) {
        if (index < list.size()) {
            Integer i = list.remove(index);
            setChanged();
            notifyObservers();
            return i;
        }
        return null;
    }
}
```

Example Code - Observer

```java
public class IntegerAdder implements Observer {
    private IntegerDataBag bag;
    public IntegerAdder(IntegerDataBag bag) {
        this.bag = bag;
        bag.addObserver(this);
    }
    public void update(Observable o, Object arg) {
        if (o == bag) {
            println("The contents of the IntegerDataBag have changed.");
            int sum = 0;
            for (Integer i : bag) {
                sum += i;
            }
            println("The new sum of the integers is: " + sum);
        }
    }
}
```

Inner Classes

- **Inner (Nested) Classes**: מחלקה פנימית היא מחלקה שאורדת תחת מחלקה אחרת (סקופ – וב המונולוג של מחלקהCHA)

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

Inner Classes

- **Inner Classes (Java)**: ב פסג' לכל מ uy יש צע茂名י המחלקה הדרומית Carl

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

Inner Classes

- **Inner Classes (Java)**: ב פסג' לכל מ uy יש צע茂名י המחלקה הדרומית Carl

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

```java
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;
        }
    }
}
```

`Height of House`
`Height of Room`

1. **Height of House**

2. **Height of Room**

3. **Same as this.height**

4. **Height of Room**

5. **Height of Room**

6. **Same as this.height**

7. **Height of Room**

8. **Height of Room**

Static Nested Classes

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

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

public class Test {
    public static void main(String[] args) {
        House.Room r = new House.Room();
        ... // Error: this room is not related to any house
    }
}

new OuterClassName.InnerClassName

---

Class - inner class
When an inner class is part of a method, it defines a limited area of visibility for that method.

- The inner class is visible only to the method containing it.
- The inner class can access and modify the method's local variables.
- The inner class cannot access or modify the method parameters.
- The inner class cannot be accessed outside the method.
- The inner class can be accessed by the method.

Example:
```java
public class Outer {
    private static class Inner implement Interface {
        ...
    }

    public static Interface getInner() {
        return new Inner();
    }
}
```

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

---

Local - inner class
- It is a class that is part of a method or an object.
- It is created when the method or object is created.
- It is destroyed when the method or object is destroyed.
- It is accessible only to the method or object it is part of.
- It can access the method's local variables.
- It cannot access the method's parameters.
- It cannot be accessed outside the method or object.

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

---

Anonymous - inner class
- It is a class that is not named.
- It is created when the method or object is created.
- It is destroyed when the method or object is destroyed.
- It is accessible only to the method or object it is part of.
- It can access the method's local variables.
- It cannot access the method's parameters.
- It cannot be accessed outside the method or object.

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

---

Singleton - inner class
- It is a class that is created only once.
- It is accessible only to the method or object it is part of.
- It can access the method's local variables.
- It cannot access the method's parameters.
- It cannot be accessed outside the method or object.

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

---

Anonymous singleton
- It is a class that is created only once.
- It is accessible only to the method or object it is part of.
- It can access the method's local variables.
- It cannot access the method's parameters.
- It cannot be accessed outside the method or object.

Example:
```java
public class Test {
    public void test(int x) {
        final int y = x + 3;
        class Info {
            public String toString() {
                return "*** " + y + " ***";
            }
        }
        Info info = new Info();
        System.out.println(info);
    }
}
```
הידור של מחלקות פנימיות

- המחלקה ( Outer$Inner.class), זורע במכ簡單 המחלקה פנימית vão שנון במחלקה הריגלה
- Outer$Inner.class הוא המחלקה הפנימית Events, שם המחלקה ישוצר
- Outer$1.class הקומפילר hềו

 Outer$1.class הקומפילר hềו