תוכנה 1 בשפת Java
שיעור חזרה

ניר אטיאס

בית הספר למדעי המחשב
אוניברסיטת תל אביב

היום בשיעור

- jPaint
- היררכיית I/O
- עוד תבניות עיצוב
  - Decorator, Visitor, Composite

jPaint

תרגיל בית

jPaint

- בתרגיל הבית נפתח את הבסיס לתוכנת ציור
- כיצד ניגשים לבניית תוכנה מורכבת?
  - מתכננים
  - נעזרים בידע קיים
  - גישת הפרד ומשול
  - בודקים

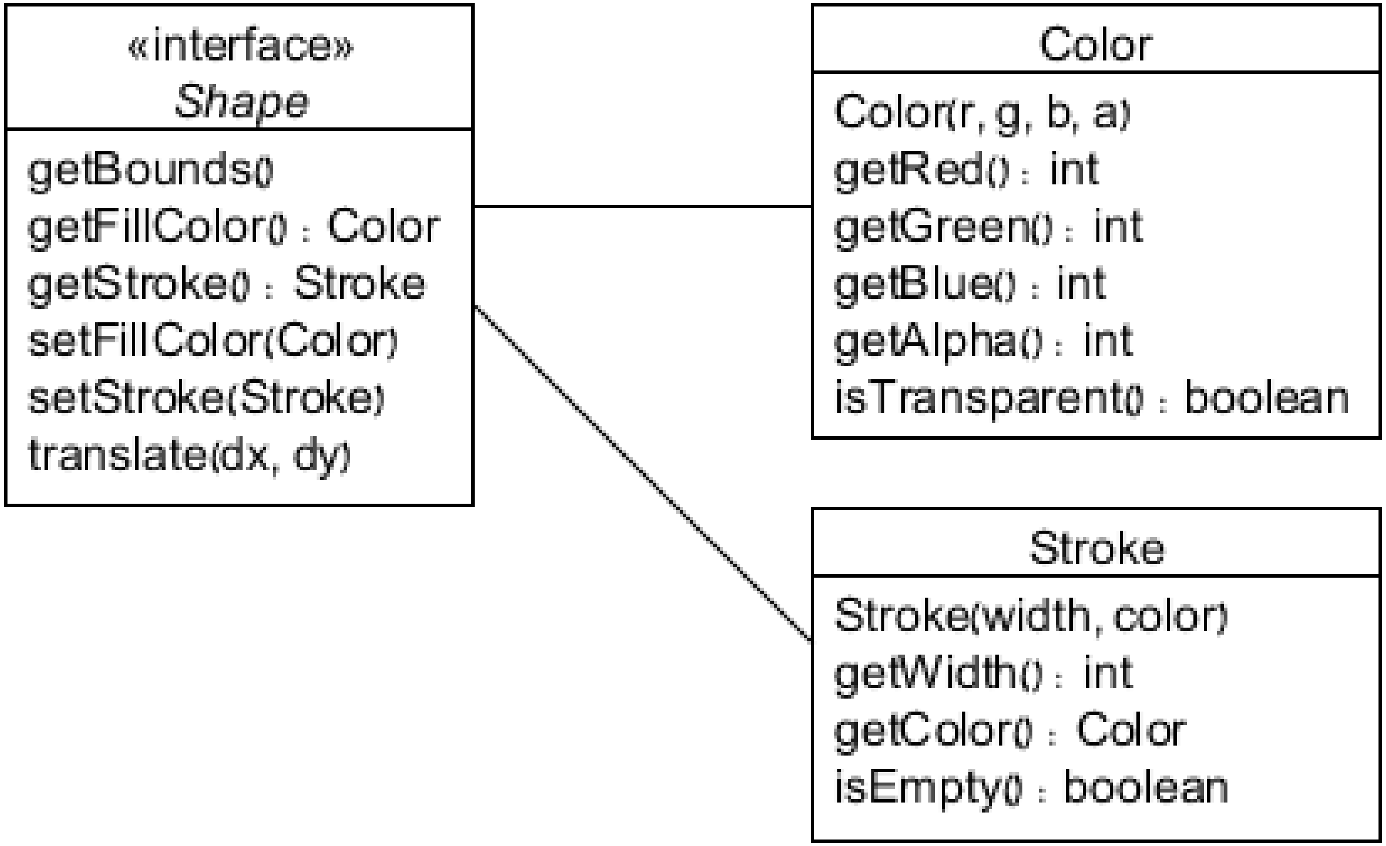
הפרדת רשויות

- נהוג להפריד תוכנה לחלקים קטנים, וחסרי תלות
- נוהג מקובל מאוד להפריד:
  - ממשק משתמש
  - לוגיקת הפעולה (לדוגמא מבני נתונים)
- ממשק המשתמש תלוי בשכבת הלוגיקה דרך מנשקים מוגדרים

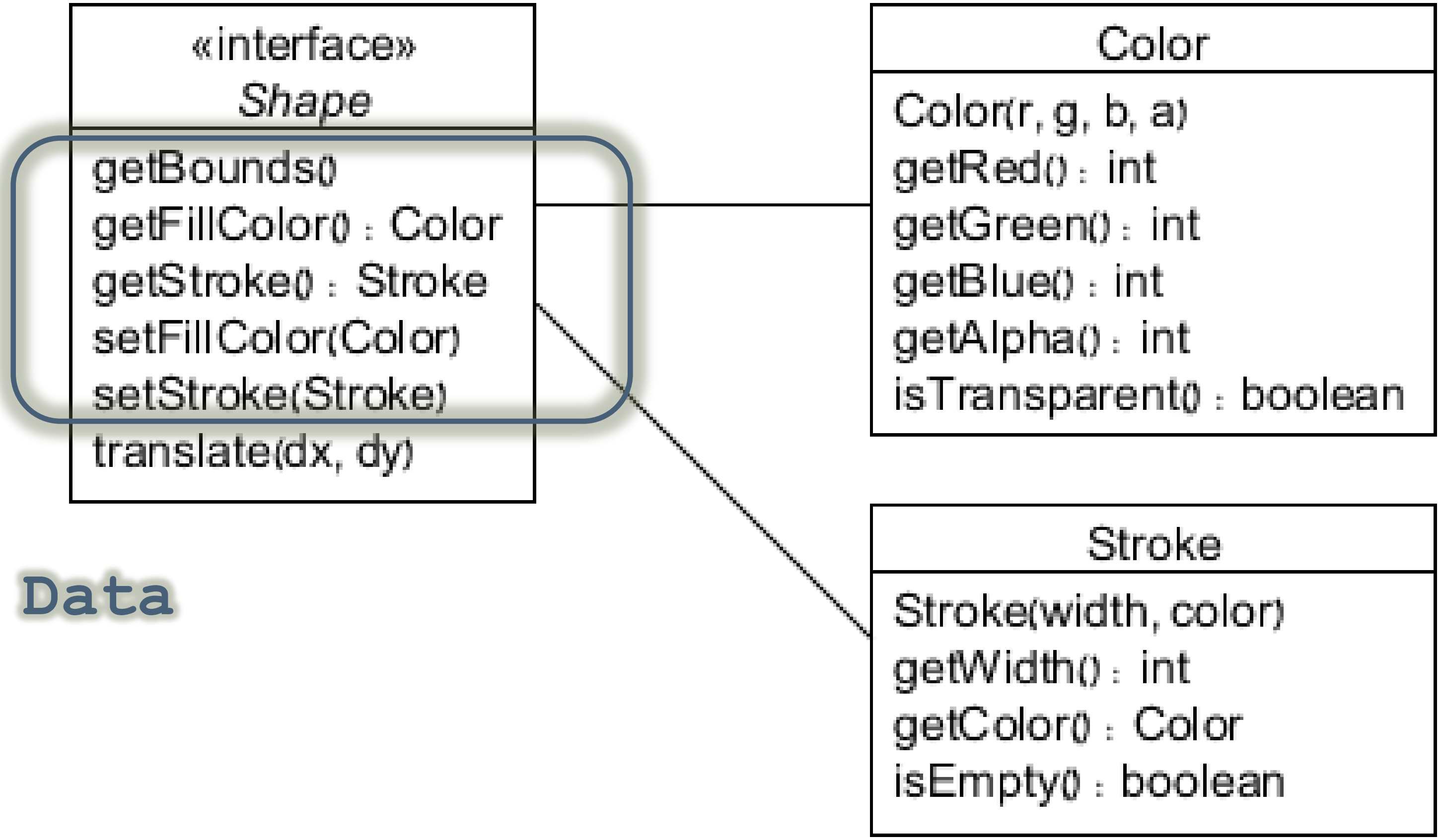
לוגיקה

- נתחיל בהגדרת מודל, כלומר בבניית מנשק למבנה נתונים (ופעולות רלוונטיות) אשר מאפשר ייצוג של ציור
- העצמים הכללים ביותר איתם אנו עובדים הן צורות.
- במה מאופיינת צורה?
  - צבע
  - מסגרת
  - מיקום בציור

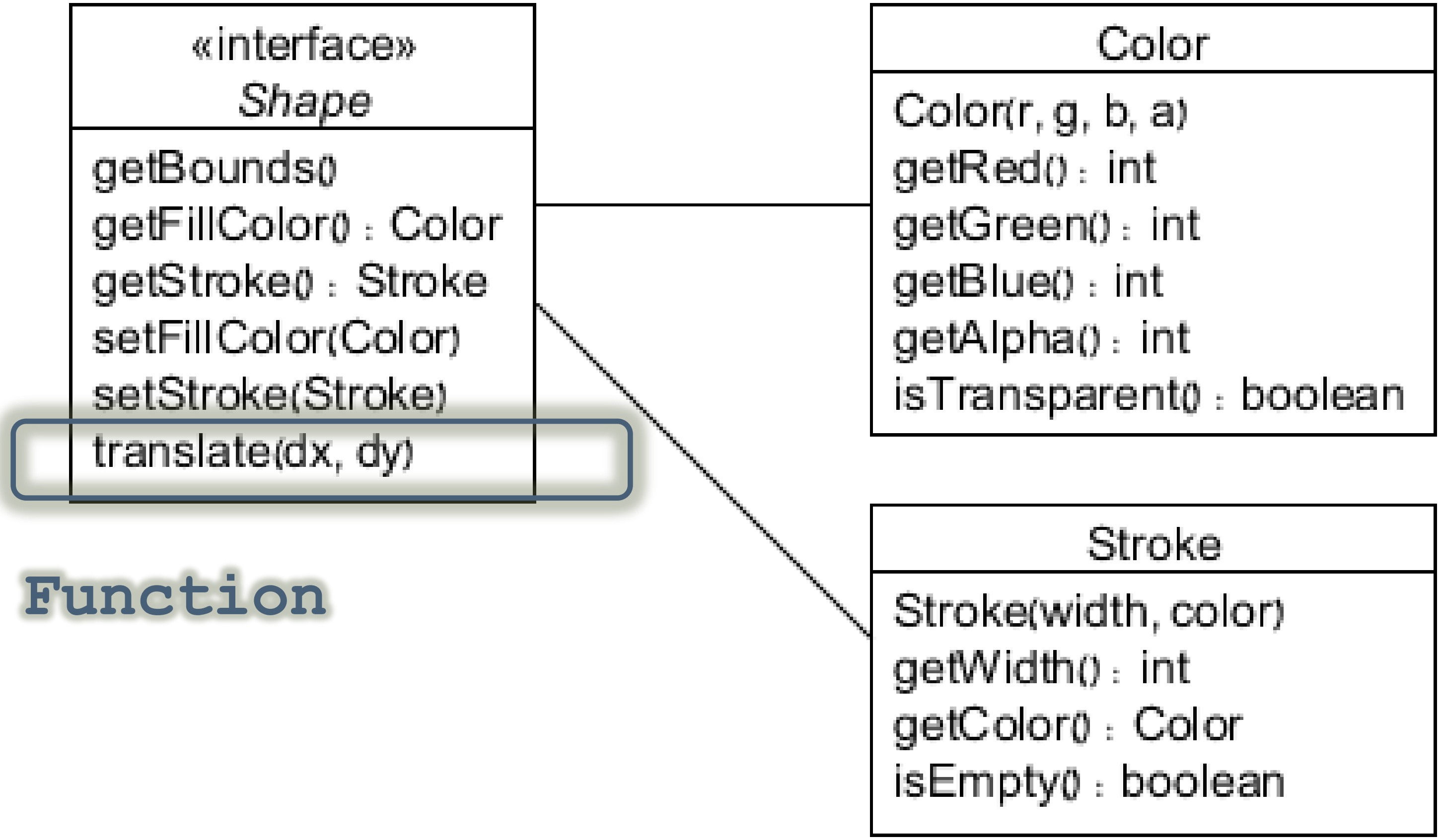
המנשק הראשון שלי



המנשק הראשון שלי

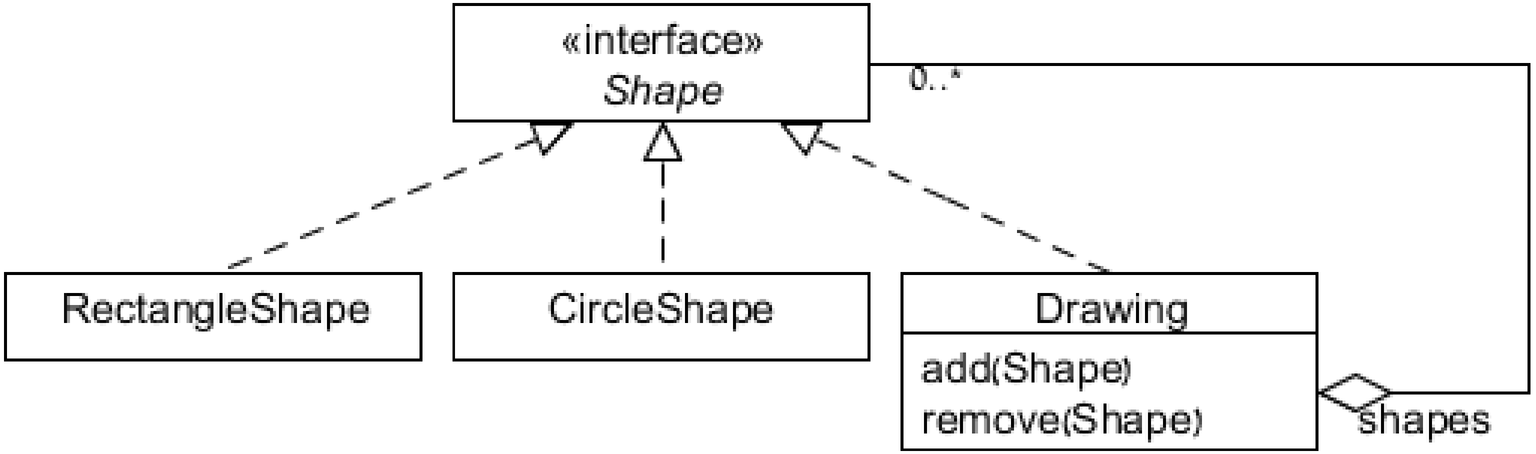


המנשק הראשון שלי



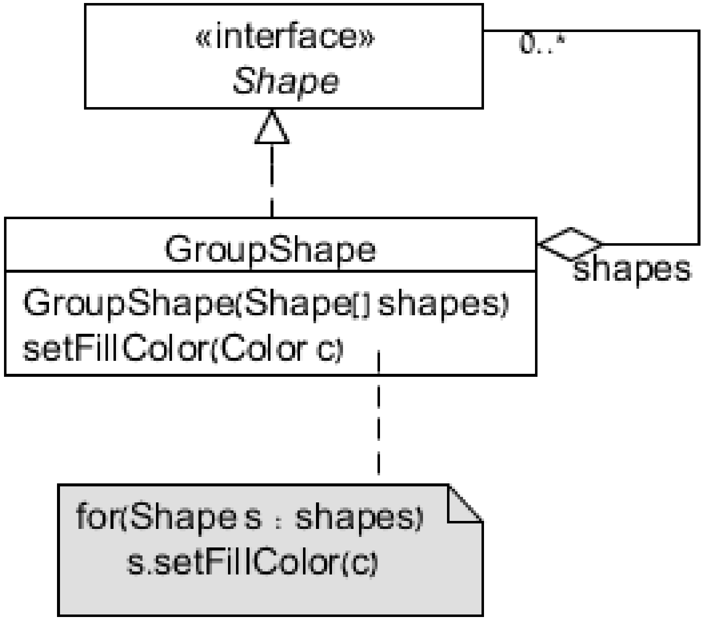
ריבוי צורות

- כעת, ניתן להוסיף את הצורות הקונקרטיות איתן נעבוד. לדוגמא:
  - ההחלטה ש-Drawing הוא Shape מאפשרת קינון של ציורים



ריבוי צורות

- ניתן גם להוסיף צורות "מיוחדות"
  - לדוגמא פעולות group ו-ungroup בתוכנות גרפיות תמומש ע"י צורה שהיא אוסף של צורות, בדומה ל- Drawing

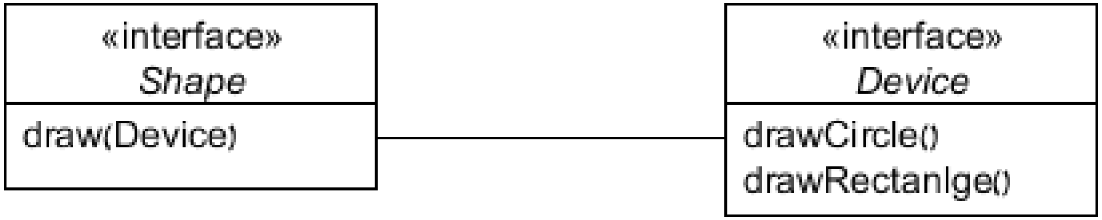


צייר לי כבשה

- נרצה ליצור תמונה ממבנה נתון של אובייקטים
- ישנן שתי בעיות:
  - כיצד מציירים?
    - על גבי חלון
      - תלות במערכת הפעלה + מערכת גרפית
    - לתוך קובץ תמונה (באיזה פורמט?)
  - טכניקת המעבר על כל האובייקטים וציורם
    - האם יש חשיבות לסדר?

כיצד מציירים?

- כרגע איננו מטפלים בממשק משתמש
  - לכן נחליט שלא מחליטים
  - נמציא מנשק למשטח ציור אשר ימומש ע"י משהו אחר
    - לדוגמא אפליקציית ציור תממש ציור ע"ג חלון
    - אפליקציית להמרת פורמטים תעביר לפורמט גרפי נתון
    - וכו' ...

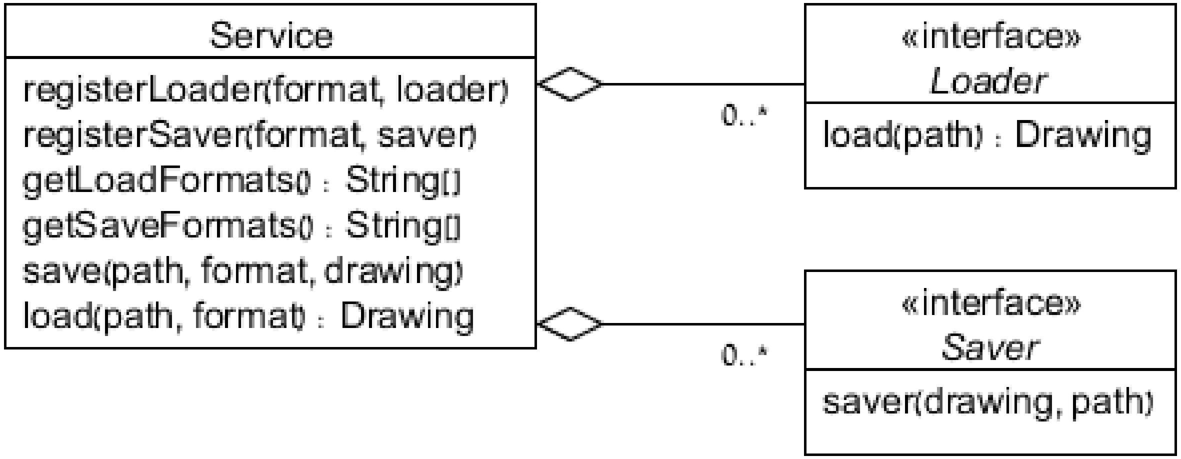


כיצד שומרים?

- כיצד ניתן להוסיף פורמט שמירה בקלות?

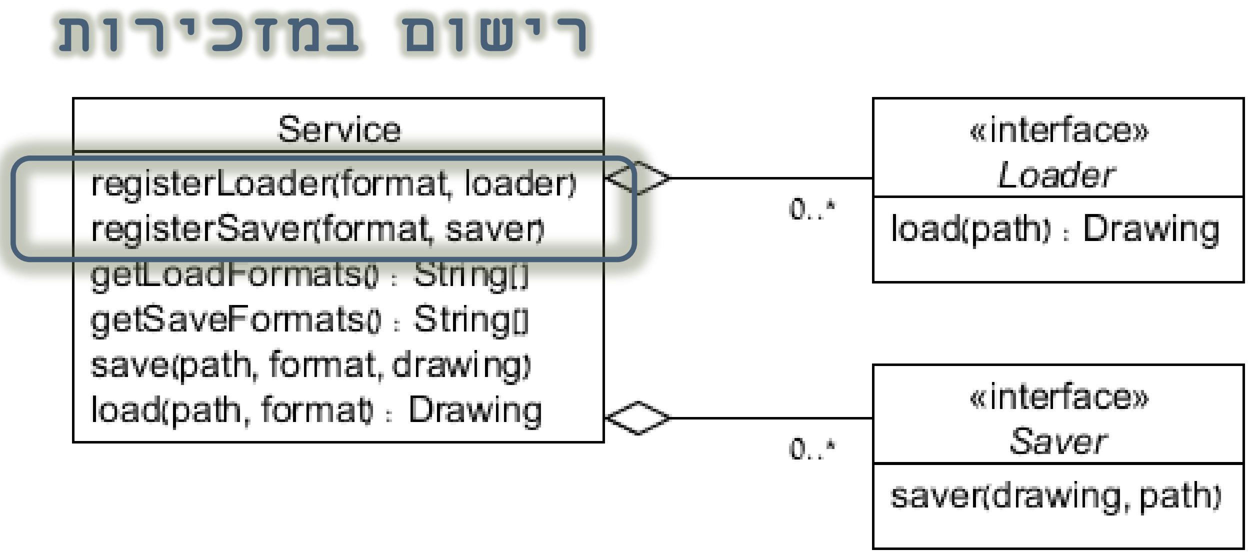
כיצד שומרים?

- כיצד ניתן להוסיף פורמט שמירה בקלות?
  - דרושה מזכירות!



כיצד שומרים?

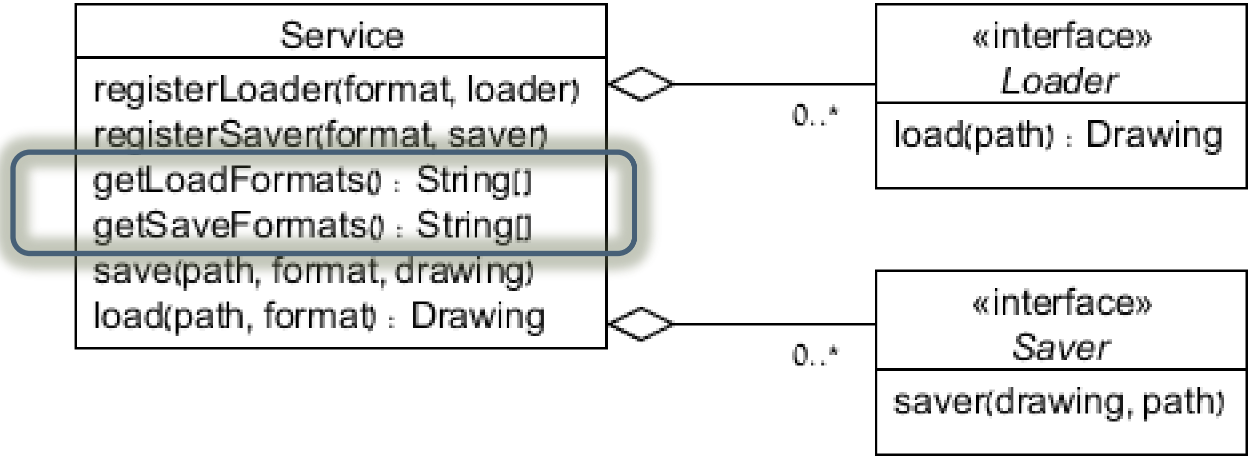
- כיצד ניתן להוסיף פורמט שמירה בקלות?
  - דרושה מזכירות!
  - מי שיודע לשמור ציורים ירשם במזכירות



כיצד שומרים?

- כיצד ניתן להוסיף פורמט שמירה בקלות?
  - דרושה מזכירות!
  - מי שיודע לשמור ציורים ירשם במזכירות
  - כאשר נרצה לשמור בפורמט מסוים נברר במזכירות אם יש מתנדבים לביצוע המשימה

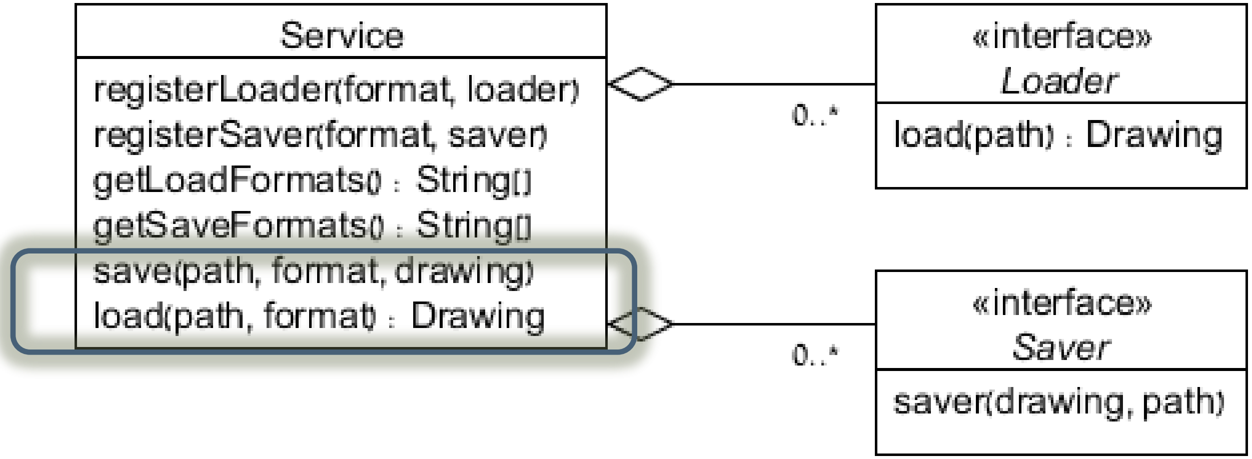
בירור



כיצד שומרים?

- כיצד ניתן להוסיף פורמט שמירה בקלות?
  - דרושה מזכירות!
  - מי שיודע לשמור ציורים ירשם במזכירות
  - כאשר נרצה לשמור בפורמט מסוים נברר במזכירות אם יש מתנדבים לביצוע המשימה

שמירה/טעינה



כיצד שומרים 2?

- כדי לממש שמירה וטעינה יש לעבור על האובייקטים
  - עבור חלק מהמימושים ניתן לייצר Device ששומר לקובץ
    - טוב עבור שמירת קבצי גרפיקה
    - לא טוב עבור שמירת המודל בכללותו
      - ייתכן שיש צורות שאינן מצוירות, כמו קבוצות
- בתרגיל – נשמור ע"י Serliaization
  - יתרונות?
  - חסרונות?

I/O

קלט פלט

רגע לפני...

- רוצים לממש תוכנה עבור ניהול כ"א בארגון.
  - העובדים מחולקים למחלקות.
  - אין מחלקות ריקות

```
/**
 * @inv getEmployees().size() >= 0
 */
public class Department {
   private List<Employee> employees;

   public List<Employee> getEmployees() {
        ...
   }
}
```

מחלקות אחרות

- הבעיה: לקוח שקורא ל-getEmployees() יכול לשנות את המשתנה employees.
- פתרון א':
  - ניתן להחזיר *עותק* של האוסף (לא יעיל)
  - לפעמים ניתן לרשום חוזה מתאים (בלתי ניתן לאכיפה)

```
/**
 * @inv getEmployees().size() >= 0
 */
public class Department {
   private List<Employee> employees;

   public List<Employee> getEmployees() {
          …
   }
}
```

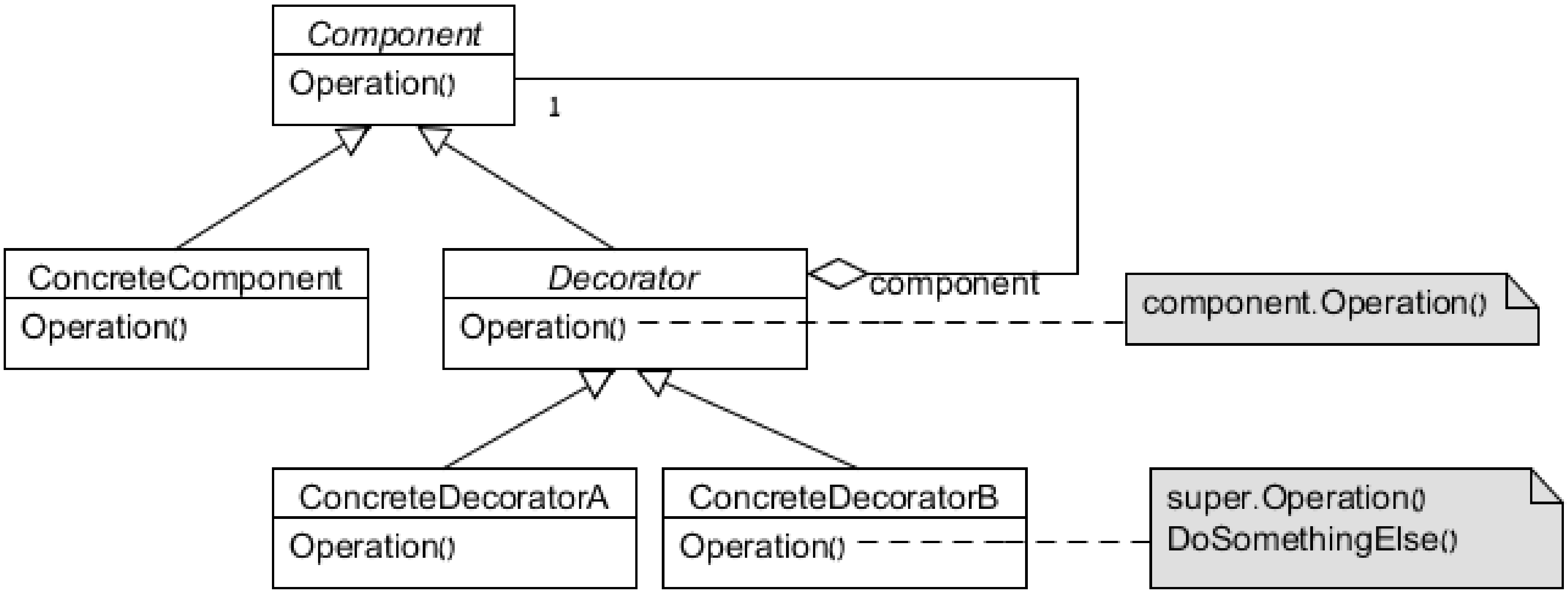
העלילה מסתבכת

- האם לכולם מותר לקרוא ל-getEmployees()?
  - לא בהכרח
  - מידע בנוגע לעובדים עלול להיות רגיש
  - ייתכן שישנם חוקים בארגון באשר לגישה למידע זה
    - הרשאות
    - תיעוד הגישות בקובץ
    - ...
  - כיצד נממש?

תבניות

- מהי התבנית החוזרת?
  - בשני המקרים אנו רוצים שמופע נתון של מחלקה יתנהג בצורה שונה ממופעים "רגילים" במחלקה
    - List שלא ניתן לשנותו
    - Department שלא מאפשרת קריאה ל-getEmployees()
  - כיצד נממש זאת?
    - תבנית העצוב Decorator

Decorator



Component
Operation()
1
ConcreteComponent
Operation()
Decorator
Operation()
component
component.Operation()
ConcreteDecoratorA
Operation()
ConcreteDecoratorB
Operation()
super.Operation()
DoSomethingElse()

ListDecorator

- עמ"נ לממש את getEmployees נפעל עפ"י תבנית העיצוב:
  - נגדיר מחלקה (אבסטרקטית) ListDecorator

```
public abstract class ListDecorator<E> implements List<E> {

   List<E> component;

   public ListDecorator(List<E> component) {
      this.component = component;
   }

   @Override
   public boolean add(E e) {
      return component.add(e);
   }

   . . .
}
```

ReadonlyList

- כעת נוכל לממש ReadonlyList

```
public class ReadonlyList<T> extends ListDecorator<T> {

    public ReadonlyList(List<T> component) {
        super(component);
    }

    @Override
    public void clear() {
        throw new RuntimeException("Read only list");
    }

    . . .
}
```

דון קישוט

- לבסוף נוכל לממש את getEmployees

```
/**
 * @inv getEmployees().size() >= 0
 */
public class Department {
   private List<Employee> employees;

   public List<Employee> getEmployees() {
       return new ReadonlyList<Employee>(employees);
   }
}
```

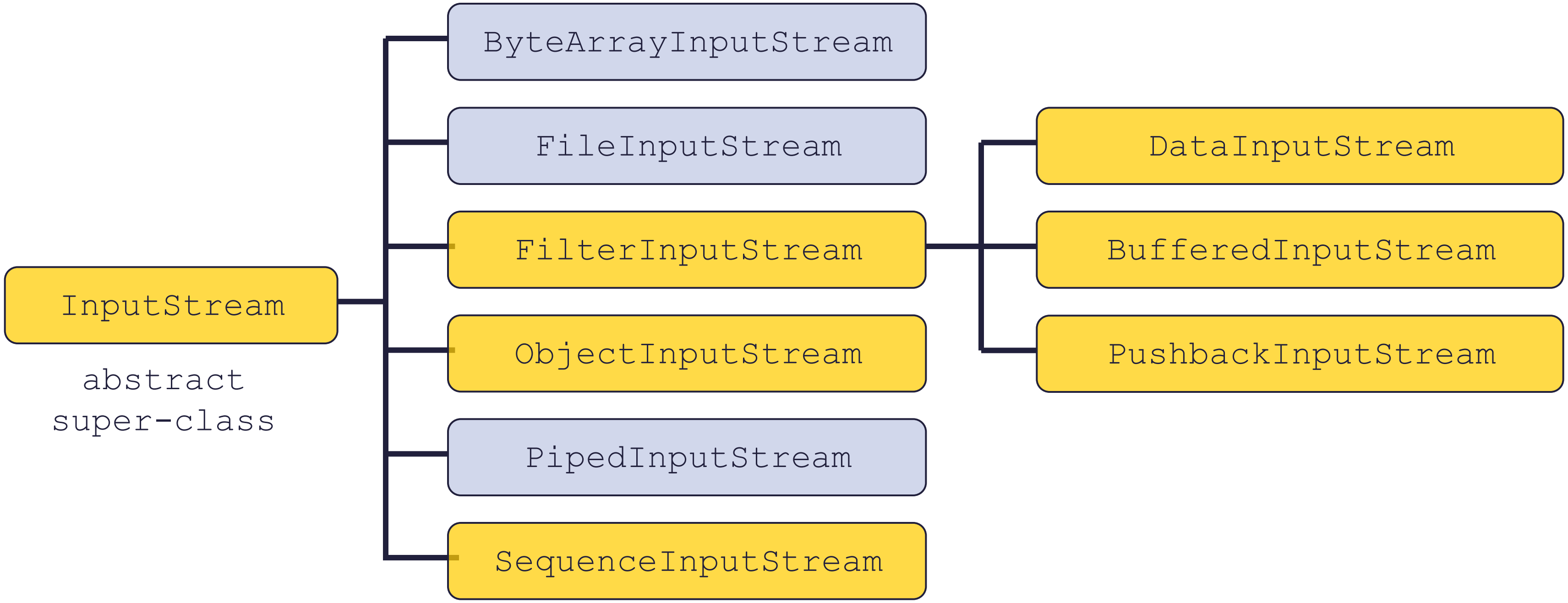
ועדת קישוט

- בדומה נוכל להגדיר Decorator למחלקה Department ולממש:
  - AccessControlDecorator
  - LogQueriesDecorator
  - ...
- בצורה זו צרכי הארגון מופרדים ממימוש נכון של מבנה הנתונים.

חזרה לקלט/פלט

- כבר ראינו את החבילה java.io
- נזכר:
  - Streams מטפלים בסדרות של נתונים
  - Readers/Writers מטפלים בקבצי טקסט
- שימוש בזרמים
  - פתיחת הזרם (בד"כ ע"י יצירת עצם מטיפוס מתאים)
  - קריאת הנתונים
  - סגירת הזרם

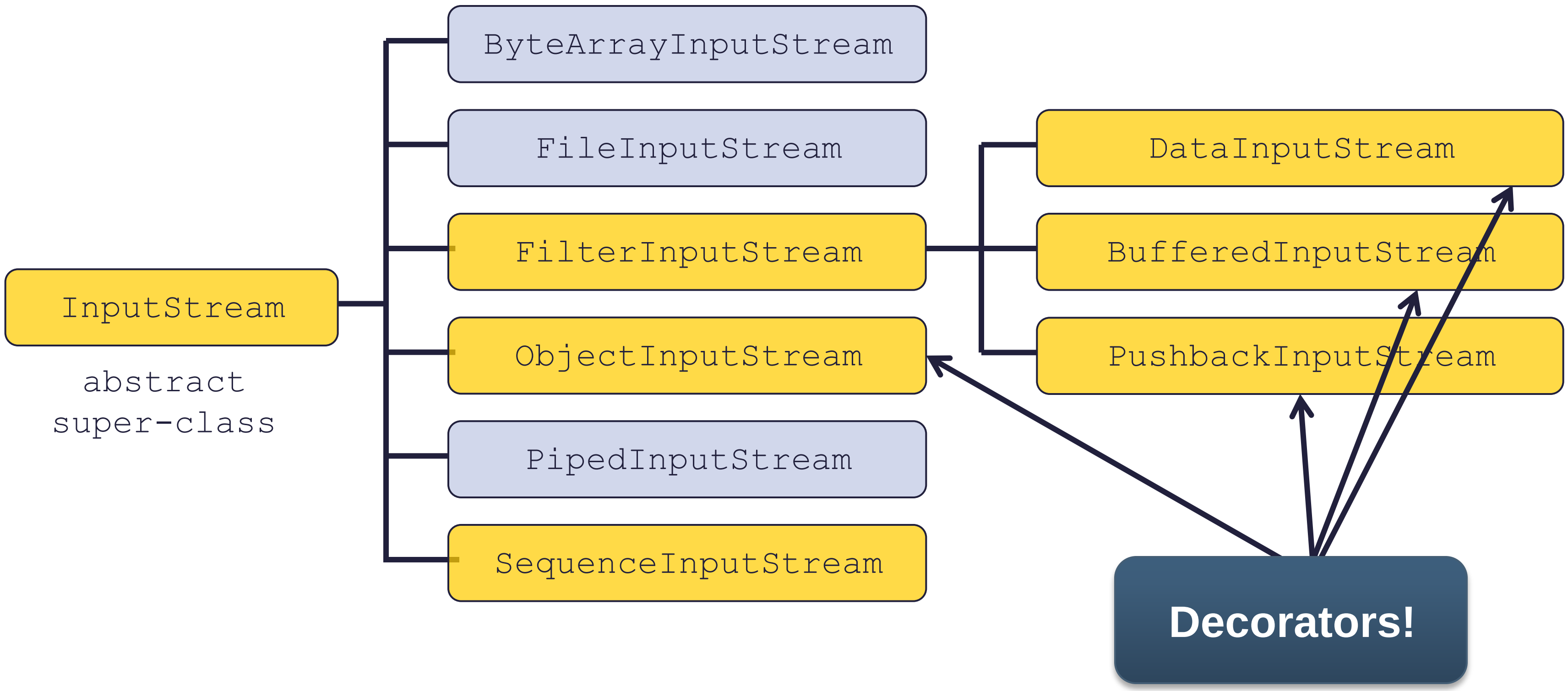
זרמי קלט - היררכיה



■ - read from data sinks

■ - perform some processing

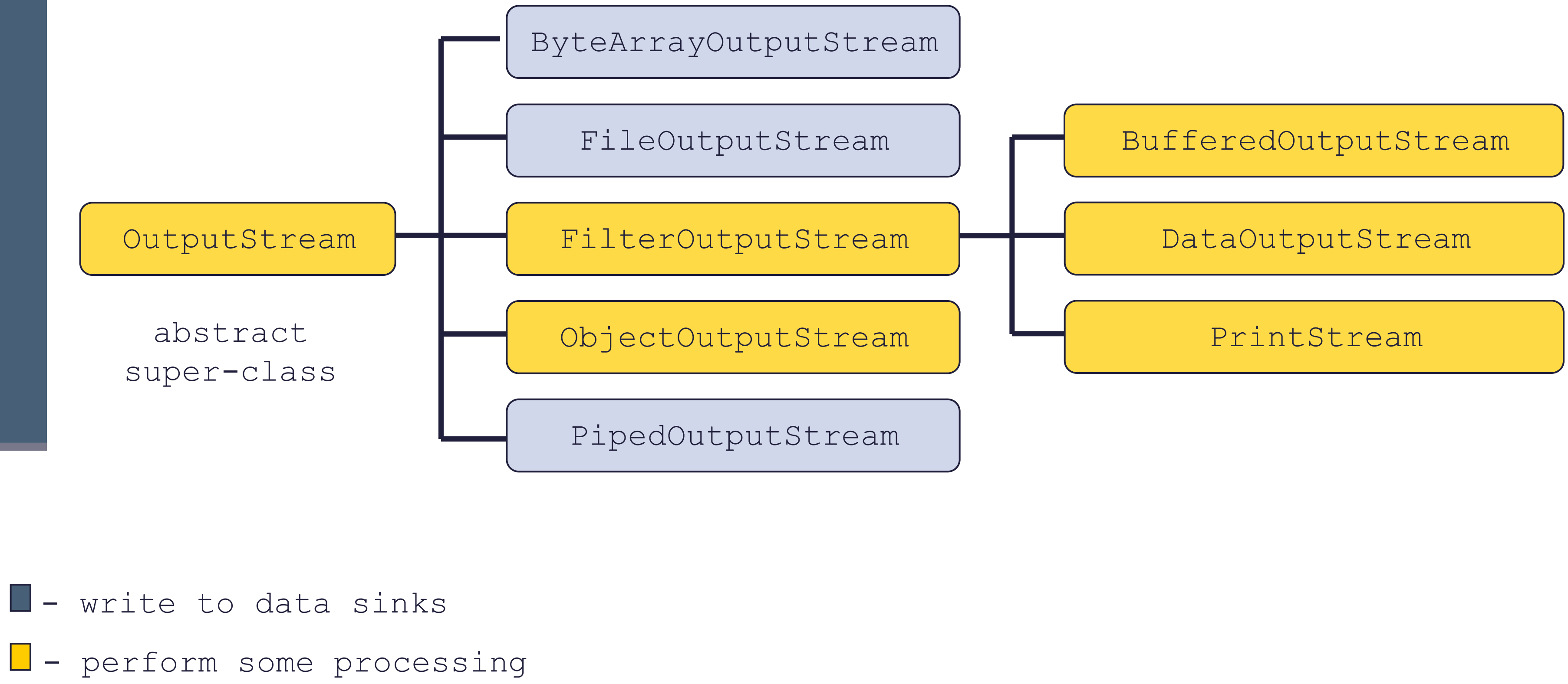
זרמי קלט - היררכיה



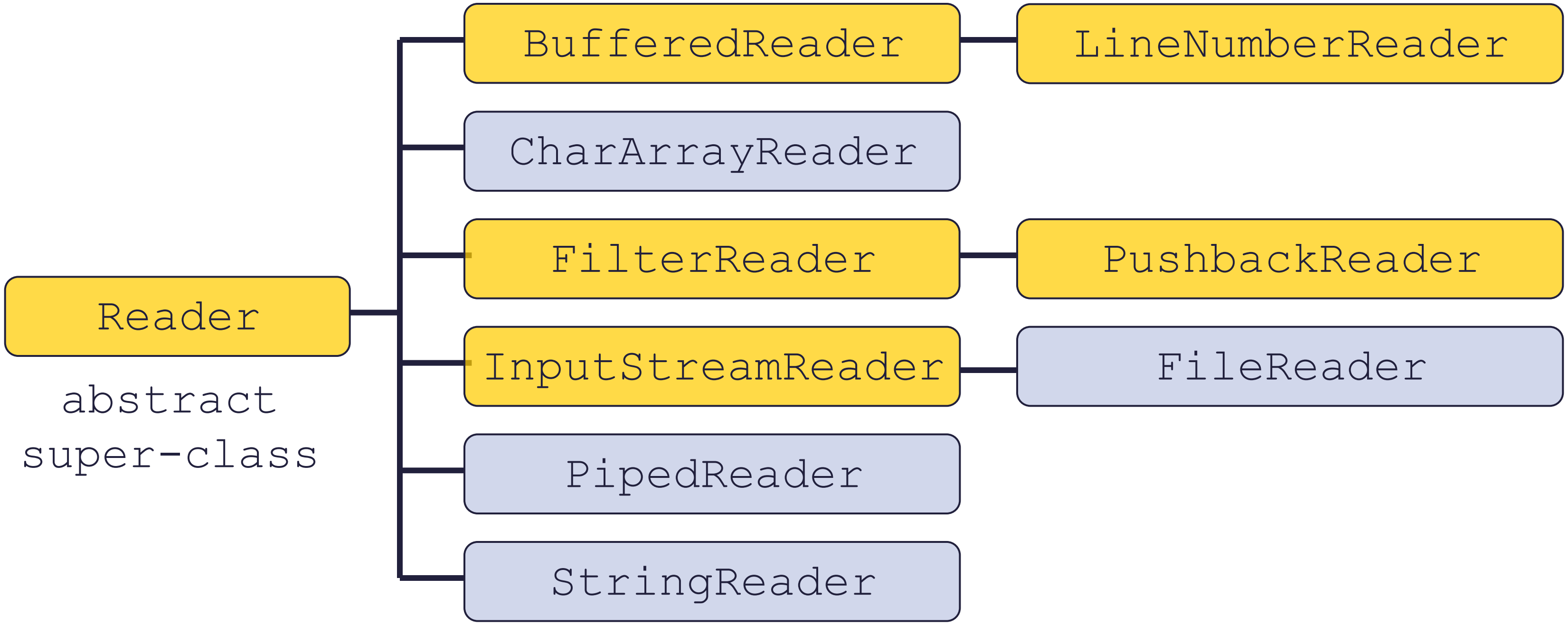
- read from data sinks

- perform some processing

היררכיית זרמי הפלט

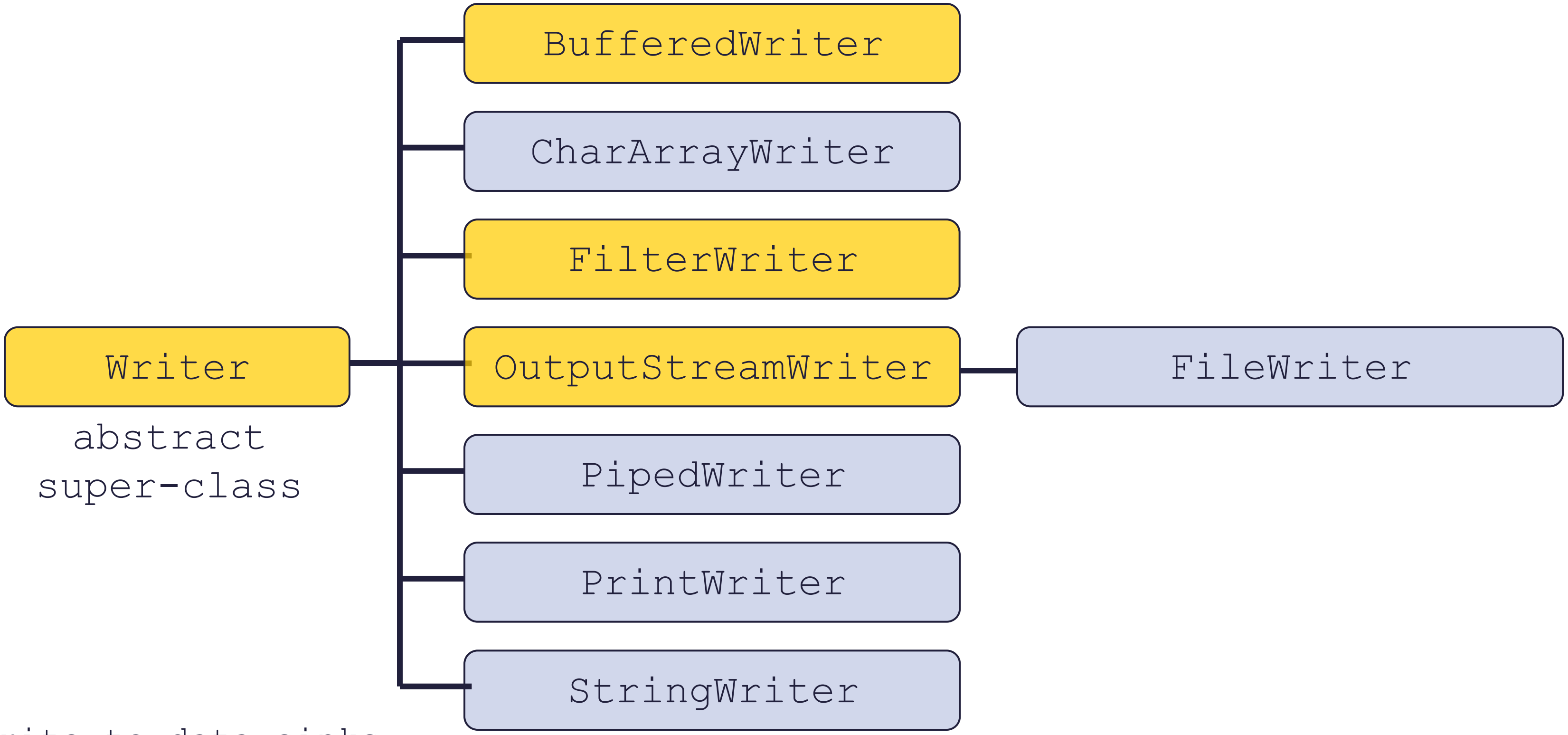


Readers



- read from data sinks
- perform some processing

Writers



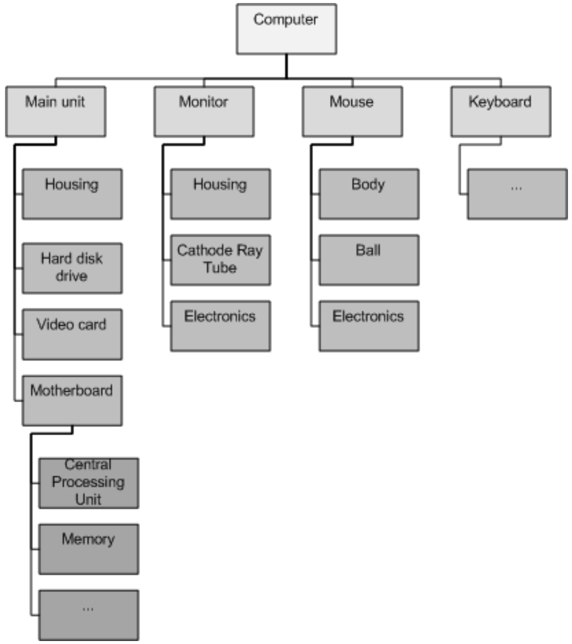
- write to data sinks

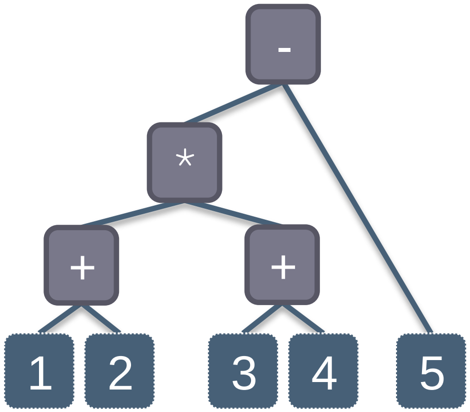
- perform some processing

תבניות עיצוב נוספות

חלק מהשלם

- פעמים רבות אנו נתקלים במבנים היררכיים:
  - ממשק משתמש (מעטפות)
  - מערכת הקבצים מחולקת לספריות
  - מוצרים מורכבים ממכלולים והם מחלקים פשוטים יותר
  - ביטויים מתמטיים:





$$(1+2)\times(3+4)-5$$

חלק מהשלם

- כחלק מחבילת תוכנה אנו נדרשים לפעול על המבנים בצורה אחידה:
  - ציור ה-UI
  - מציאת קבצים
  - קביעת העלות של מוצר
  - שערוך הביטוי המתמטי

דוגמא: שערוך

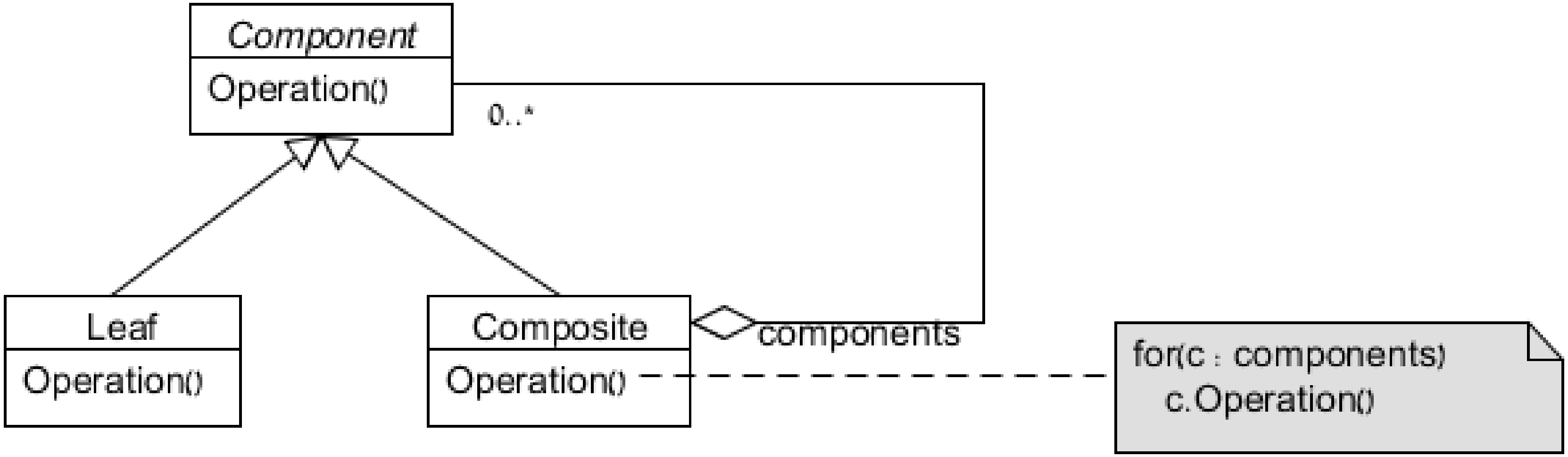
```
public static double evaluate(Expression e)  {
    if(e instanceof BinaryOperator) {
        if(e instanceof Plus) {
            Plus p = (Plus)e;
            return evaluate(p.getLhs()) + evaluate(p.getRhs());
        } else if (e instanceof Minus) {
            Minus p = (Minus)e;
            return evaluate(p.getLhs()) - evaluate(p.getRhs());
        } else if (e instanceof Multiply) {
            Multiply p = (Multiply)e;
            return evaluate(p.getLhs()) * evaluate(p.getRhs());
        }
    } else if (e instanceof ConstantExpression) {
        return ((ConstantExpression)e).getValue();
    }

    throw new RuntimeException("Unknown operation");
}
```

שערוך

- המימוש שביר
  - בכל שינוי לביטויים האפשריים יש לשנות את הפונקציה
    - תמיכה באופרטורים נוספים
      - העלאה בחזקה
      - שורש
      - לוגריתם
      - ...
  - קל לשכוח להוסיף מימוש לאופרטורים חדשים

תבנית העיצוב Composite

- תבנית העיצוב Composite נותנת מענה לבעיה.



מימוש השערוך לפי Composite

- נגדיר מנשק עם הפעולה הרצויה

```
public interface Expression {
    public double evaluate();
}
```

- קל לשערך ביטויים קבועים (עלים):

```
public class ConstantExpression implements Expression {
    double value;

    public ConstantExpression(double value) {
        this.value = value;
    }
    @Override
    public double evaluate() {
        return value;
    }
}
```

מימוש השערוך לפי Composite

- במקרה שלנו ה-Composite (לפחות כרגע) פשוטים:

```
public abstract class BinaryOperator implements Expression {

    private Expression lhs;
    private Expression rhs;

    public BinaryOperator(Expression lhs, Expression rhs) {
        this.lhs = lhs;
        this.rhs = rhs;
    }
    @Override
    public double evaluate() {
        return op(lhs.evaluate(), rhs.evaluate());
    }

    protected abstract double op(double lhs, double rhs);
}
```

```java
public class Plus extends BinaryOperator {

    public Plus(Expression lhs, Expression rhs) {
        super(lhs, rhs);
    }
    @Override
    protected double op(double lhs, double rhs) {
        return lhs + rhs;
    }
}

public class Minus extends BinaryOperator {

    public Minus(Expression lhs, Expression rhs) {
        super(lhs, rhs);
    }
    @Override
    protected double op(double lhs, double rhs) {
        return lhs - rhs;
    }
}
```

- והשימוש, פשוט:

```
public static void main(String[] args) {
   Expression e = new Minus(
     new Multiply(
        new Plus(new ConstantExpression(1),
                 new ConstantExpression(2)),
        new Plus(new ConstantExpression(3),
                 new ConstantExpression(4))),
     new ConstantExpression(5));

   System.out.println(e.evaluate());
}
```

- בתרגיל הבית המתודה Draw פועלת בדיוק לי תבנית העיצוב Composite

Here to stay

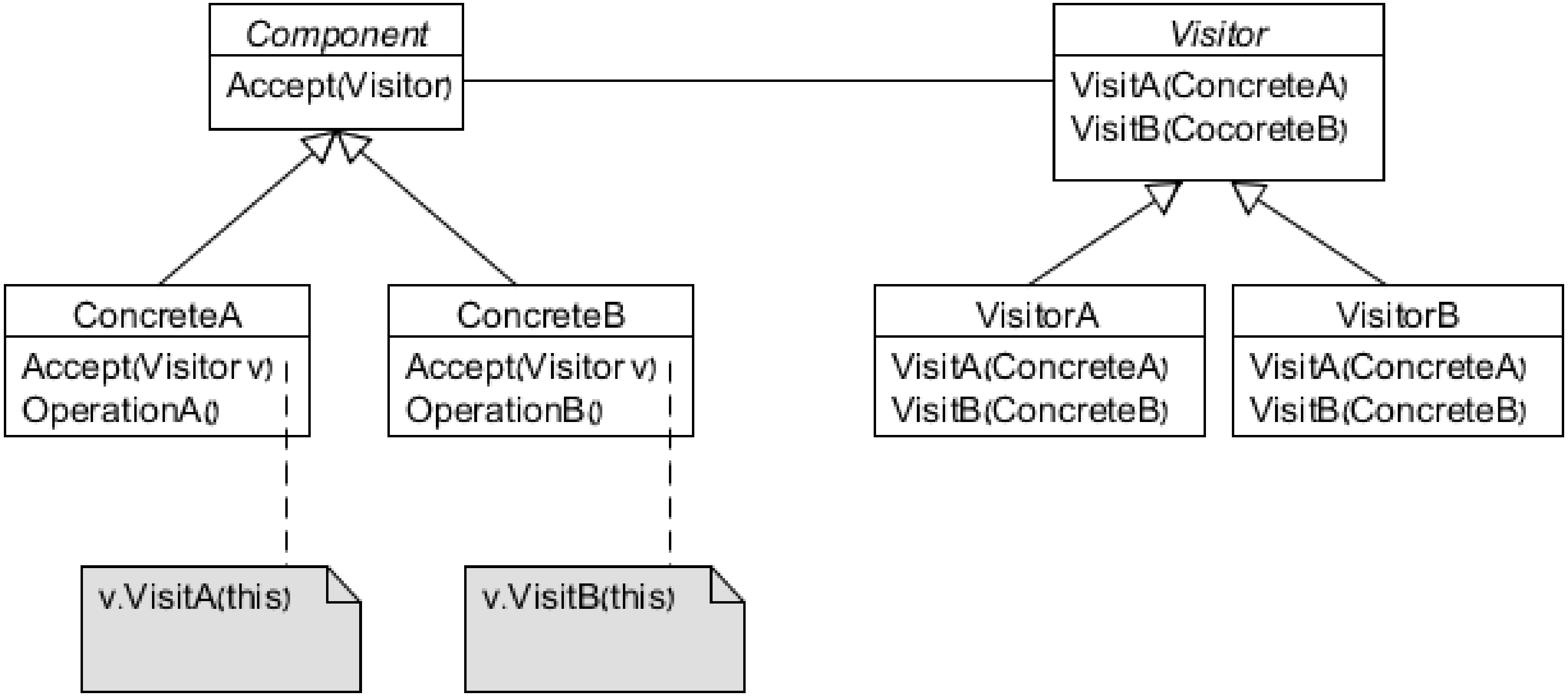
- פעולה שכיחה על מבנה הנתונים היא שמירה לקובץ.
- כיצד נבצע זאת?
  - נעבור על מבנה הנתונים
  - נזהה את סוג העצם ונשמור אותו לזרם
- ניתן להשתמש בתבנית בעיצוב Composite גם כאן:

```
public interface Expression {
    public double evaluate();
    public void save(Writer stm);
}
```

שאלות

- כיצד נתמוך במספר רב של פורמטים?
  - הרבה מתודות מגדילות את המחלקה ומסבכות את תחזוקתה
  - המתודות נקבעות בצורה סטטית בזמן הקומפילציה
- לפנינו בעיה מעט שונה מהבעיה הקודמת:
  - העץ נתון (ונניח כי מבנהו יציב)
  - האלגוריתמים הפועלים עליו מגוונים
  - אנו נדרשים לזהות את הסוג של מרכיביו

תבנית העיצוב Visitor



מימוש בביטויים מתמטיים

- נרחיב את המנשק Expression:

```
public interface Expression {
    public double evaluate();
    public void postorder(ExpressionVisitor v);
}
```

- נגדיר את ExpressionVisitor:

```
public interface ExpressionVisitor {
    public void visitConstant(ConstantExpression e);
    public void visitPlus(Plus p);
    public void visitMinus(Minus m);
    public void visitMultiply(Multiply m);
    public void visitPower(Power p);
}
```

המרה לקוד Java

```
public class JavaCodeVisitor implements ExpressionVisitor {
    Stack<String> stack = new Stack<String>();

    protected void pushOperator(String op) {
        String last = stack.pop();
        String first = stack.pop();
        stack.push("(" + first + op + last + ")");
    }

    public void visitConstant(ConstantExpression e) {
        stack.push(String.valueOf(e.getValue()));
    }

    public void visitPlus(Plus p) {
        pushOperator("+");
    }
    ...
    public String getExpression(String name) {
        return "double " + name + " = " + stack.peek();
    }
}
```

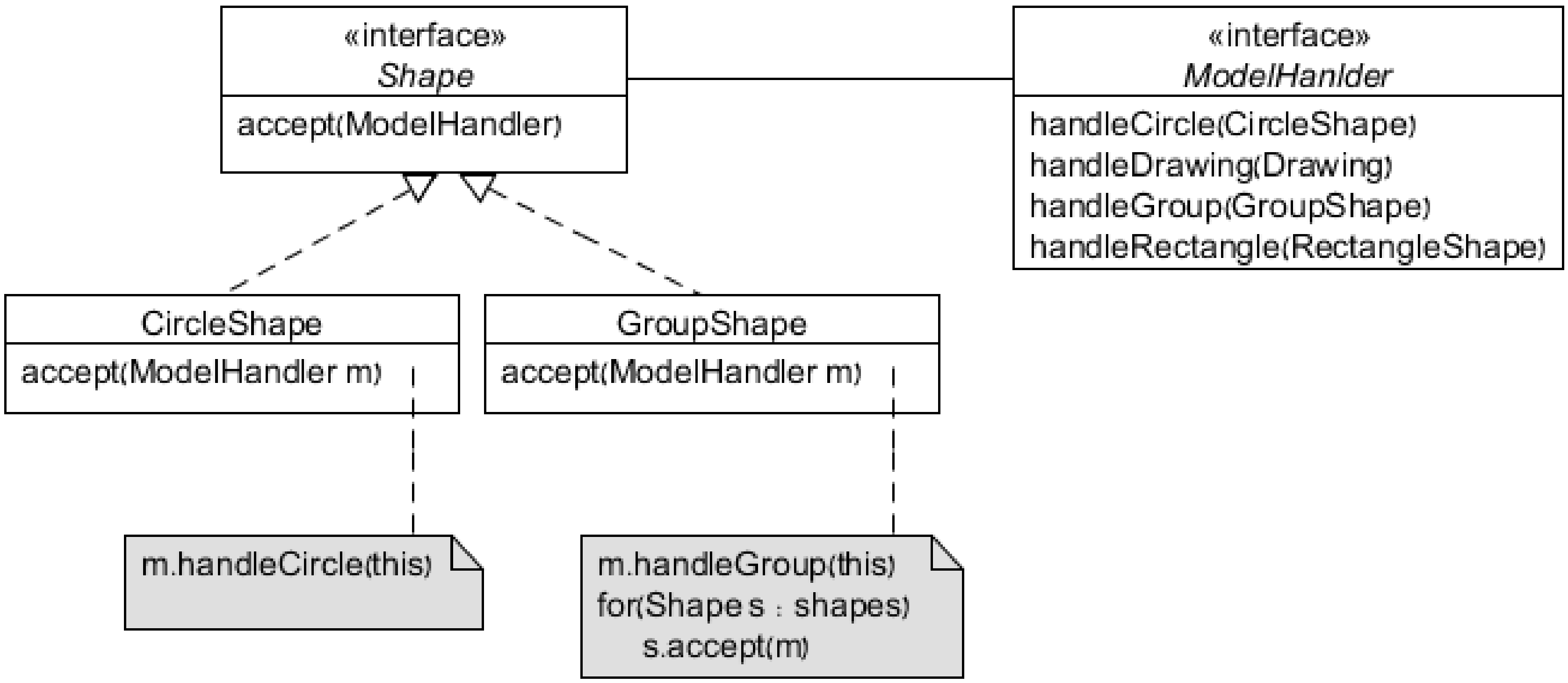
והשימוש:

```java
public static void main(String[] args) {
   Expression e = new Minus(
     new Multiply(
        new Plus(new ConstantExpression(1),
                 new ConstantExpression(2)),
        new Plus(new ConstantExpression(3),
                 new ConstantExpression(4))),
     new ConstantExpression(5));

   JavaCodeVisitor vis = new JavaCodeVisitor();
   e.postorder(vis);

   System.out.println(vis.getExpression("e"));
}
```

```
double e = (((1.0+2.0)*(3.0+4.0))-5.0)
```

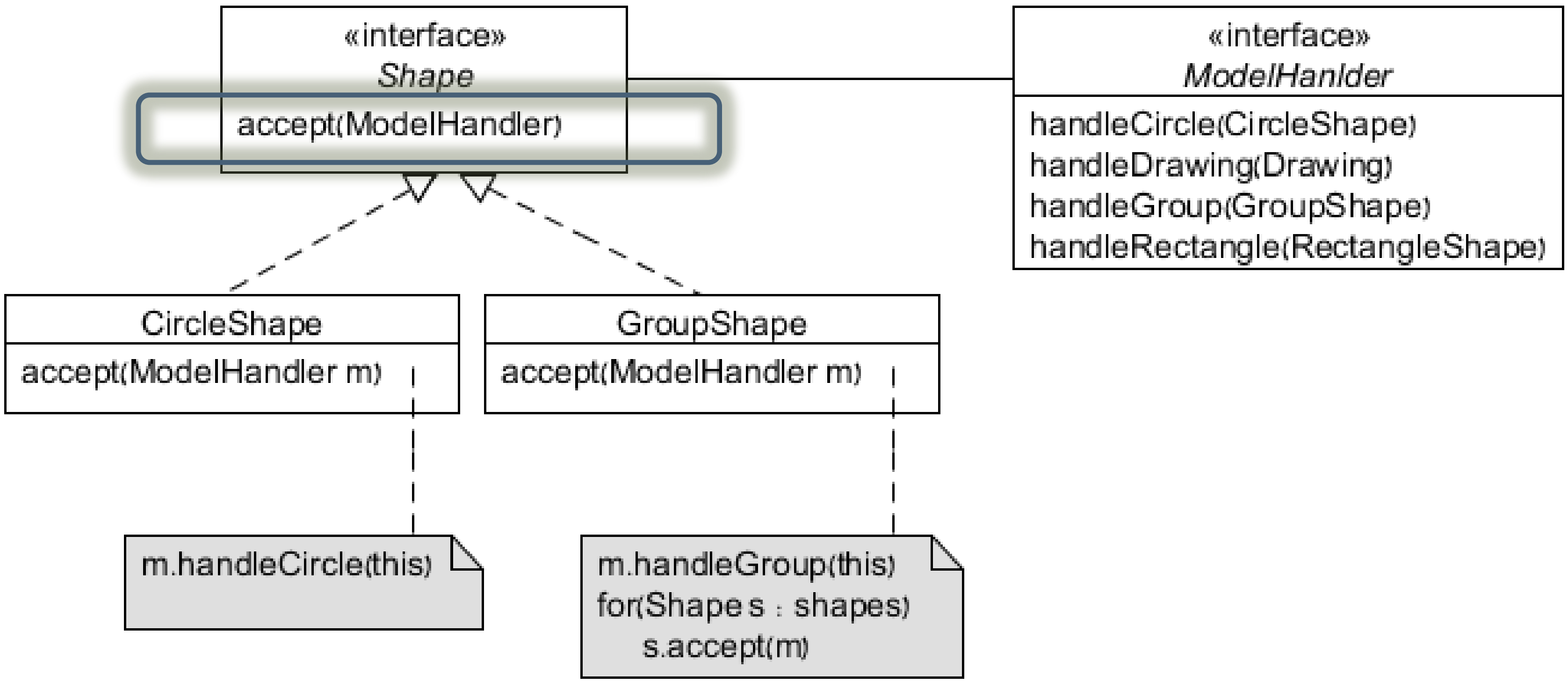
דוגמא נוספת: jPaint

- בתרגיל הבית, ניתן היה לממש שמירה גם כך:

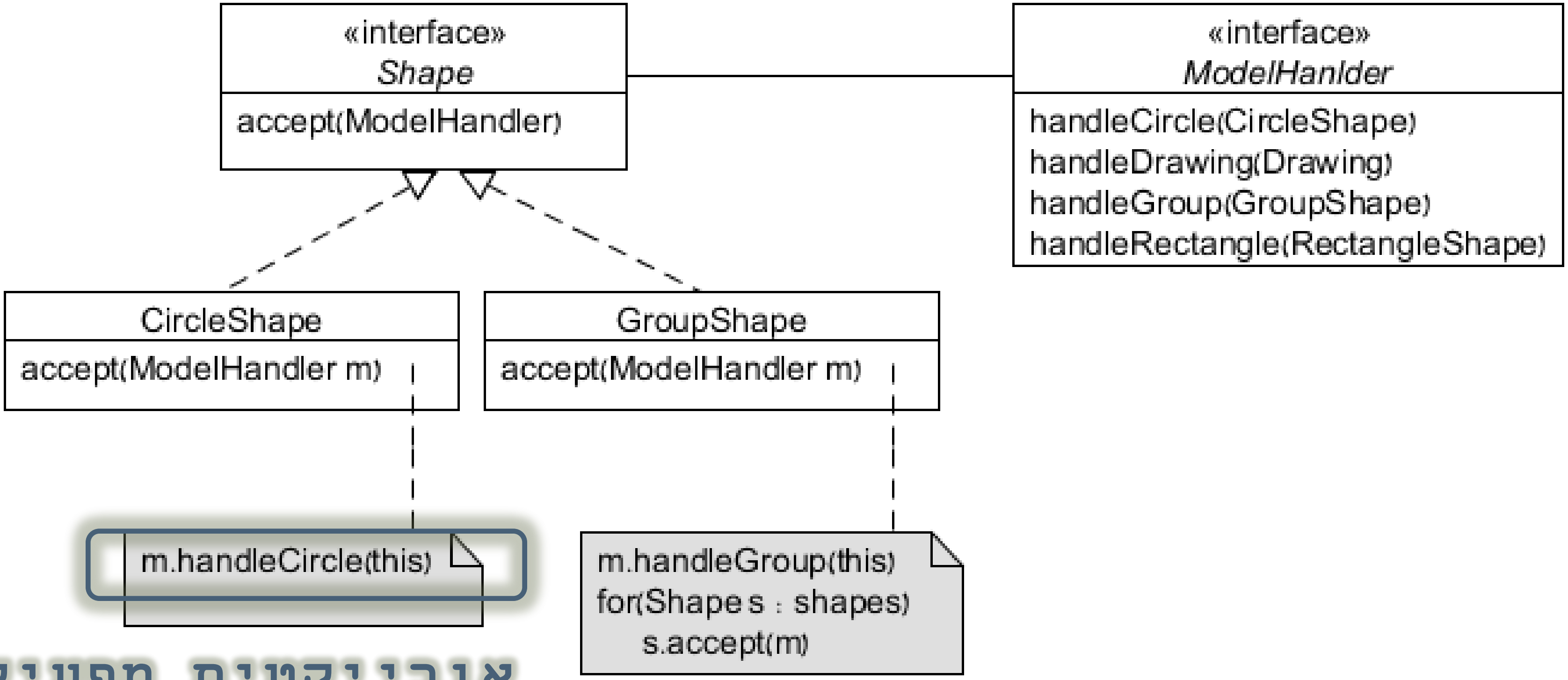


שמירה

מעבר כללי על המודל

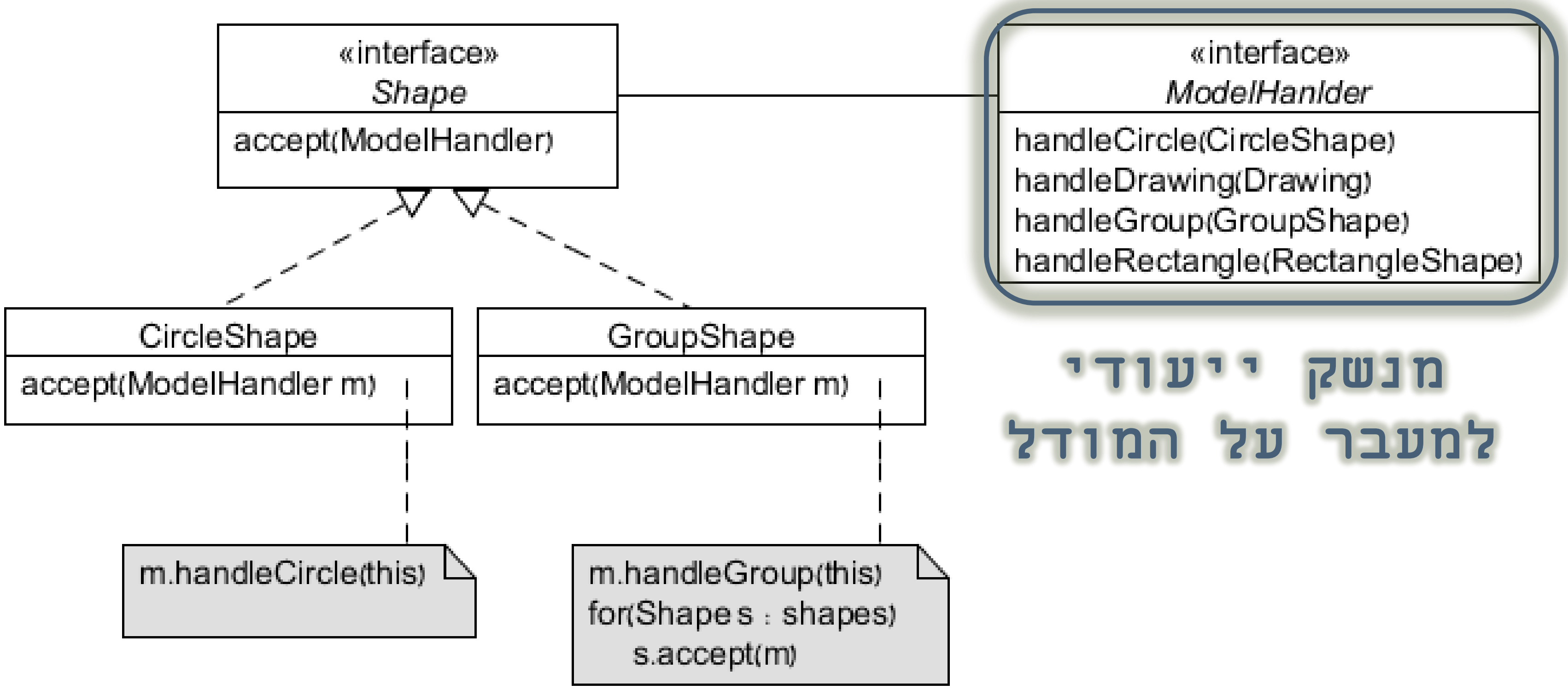


שמירה



אובייקטים מפעילים פונקציות יעודיות

שמירה



מנשק ייעודי
למעבר על המודל