

# תכנות מונחה עצמים בשפת C++

אוהד ברזילי

אוניברסיטת תל אביב

# עיצוב מחלקות דוגמא מודרכת

המצגת מכילה קטעים מתוך מצגת של פרופ' עמירם יהודאי  
ע"פ הספר:

Object-Oriented Software Construction, 2nd edition,  
by Bertrand Meyer (Prentice Hall) .

כל הזכויות שמורות למחברים

# הערות על טיפוסים

- הפרה של כללי הטיפוס קורת כאשר עבור הקריאה  $x.f(arg)$ , כש- $x$  מתייחס לעצם OBJ:
  - אין ל OBJ התכונה  $f$  (או שהיא אינה זמינה עבורו)
  - התכונה קיימת אך הטיפוס של  $arg$  אינו מתאים עבורה ואין לו המרה (casting) לטיפוס מתאים
- הפרה של כללי טיפוס ניתן למצוא סטטית (טעות קומפילציה) או דינאמית (בזמן ריצה)

# הערות על טיפוסים

■ הבעיה מורכבת משני חלקים:

– בעיית הטיפוס (typing) – האם בזמן ריצה תהיה קיימת תכונה f זמינה עבור OBJ?

– בעיית הקישור (binding) – איזו תכונה תתבצע בפועל?

■ ב C++ ברירת המחדל היא קישור סטטי, מתודות virtual יוצרות קישור דינאמי

■ בדיקת הטיפוס ב C++ היא תמיד סטטית

# הערות על טיפוסים

■ ראינו במהלך הקורס שתי בעיות שיוצרת בדיקת טיפוסים סטטית:

– בעיית השונות המשותפת (covariant), הגדרה מחדש של טיפוס הארגומנטים של מתודות)

– הסתרת תכונות – Descendant hiding - הפיכה של תכונה ציבורית נורשת לפרטית

# דוגמא מודרכת

## עיצוב מערכת עם לוחות מרובים

### - Enquiry on Flights -

Flight sought from:  To:

Departure on or after:  On or before:

Preferred airline (s):

Special requirements:

---

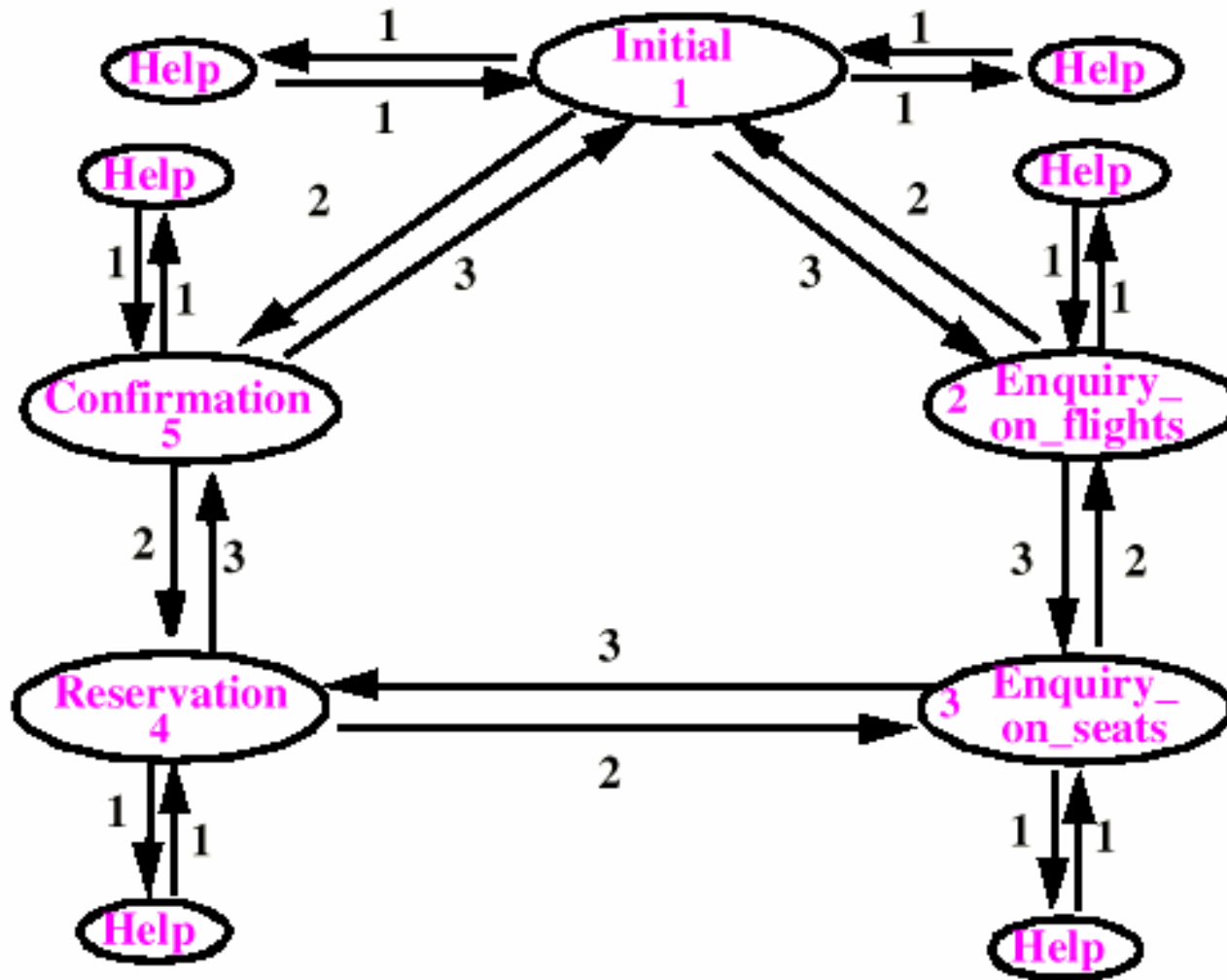
AVAILABLE FLIGHTS: 1

**Flt# AA 42      Dep 8:25      Arr 7:45      Thru: Chicago**

Choose next action:

- 0 — **Exit**
- 1 — **Help**
- 2 — **Further enquiry**
- 3 — **Reserve a seat**

# המערכת כאוטומט מצבים



תכנות מונחה עצמים בשפת C++  
אוניברסיטת תל אביב

# נסיון ראשון

*Enquiry:*

“Display *Enquiry on flights* panel”

**repeat**

“Read user’s answers and choice *C* for the next step”

**if** “Error in answer”

**then** “Output appropriate message” **end**

**until not** “error in answer” **end**

“Process answer”

**switch** (*C*)

{

*case C<sub>0</sub>* : **goto** *Exit*

*case C<sub>1</sub>* : **goto** *Help*

*case C<sub>2</sub>* : **goto** *Reservation*

...

}

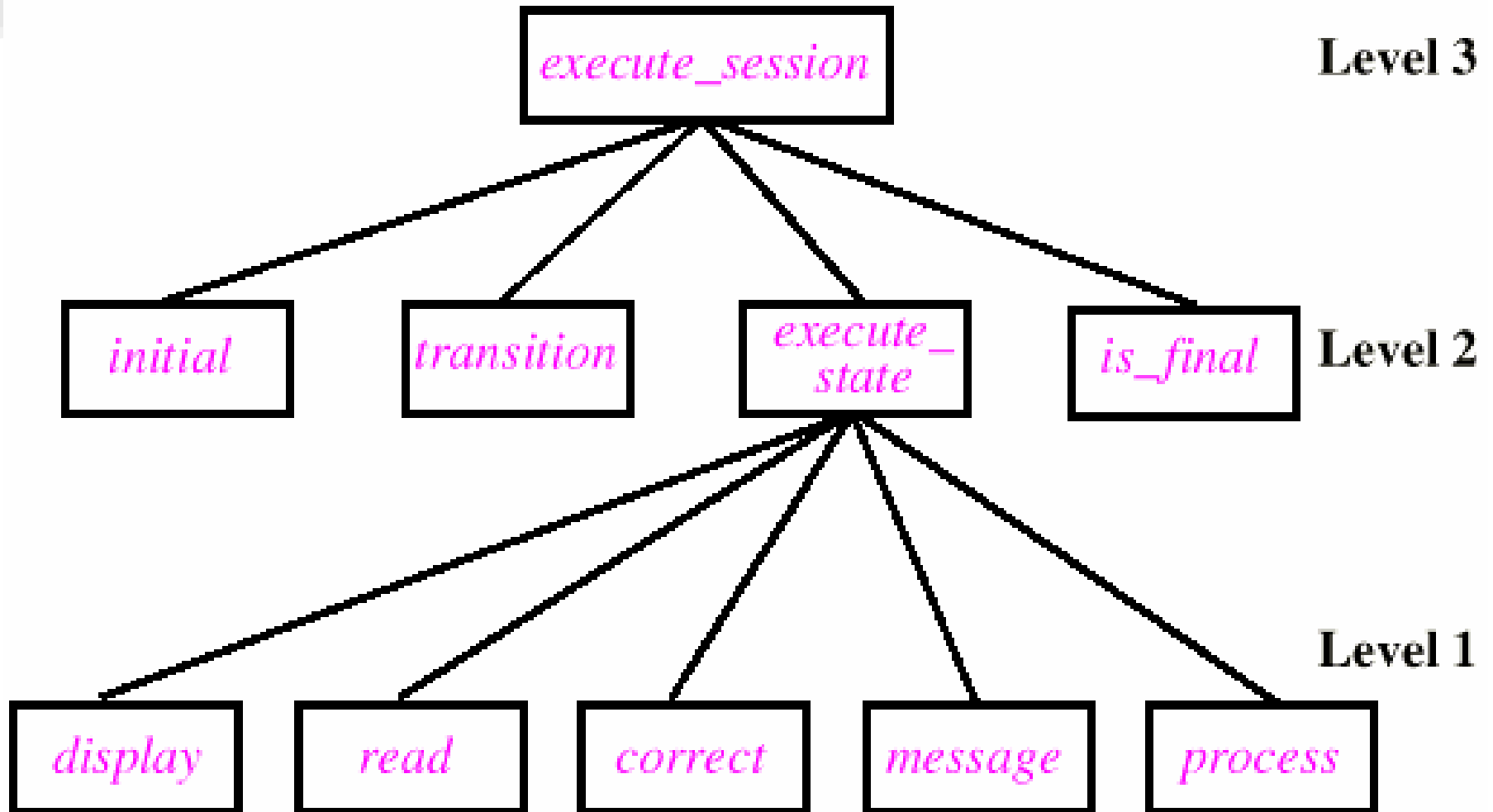




# טבלת מעבר מצבים

Choice → ↓ State	0	1	2	3
<b>1</b> ( <i>Initial</i> )	<b>-1</b>	<b>0</b>	<b>5</b>	<b>2</b>
<b>2</b> ( <i>Flights</i> )		<b>0</b>	<b>1</b>	<b>3</b>
<b>3</b> ( <i>Seats</i> )		<b>0</b>	<b>2</b>	<b>4</b>
<b>4</b> ( <i>Reserv.</i> )		<b>0</b>	<b>3</b>	<b>5</b>
<b>5</b> ( <i>Confirm</i> )		<b>0</b>	<b>4</b>	<b>1</b>
<b>0</b> ( <i>Help</i> )		<i>Return</i>		
<b>-1</b> ( <i>Final</i> )				

# פירוק פונקציונלי top-down



# execute\_session()

```
/** Execute a complete session of the interactive system
 */
void execute_session()
{
    int state, choice;

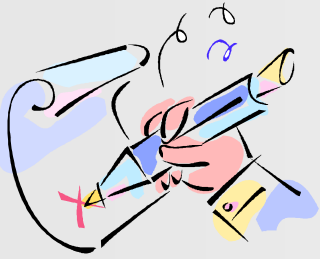
    state = initial();

    while(!is_final(state))
    {
        execute_state(state, choice);
        // Routine execute_state updates value of choice

        state = transition(state, choice);
    }
}
```

# execute\_state()

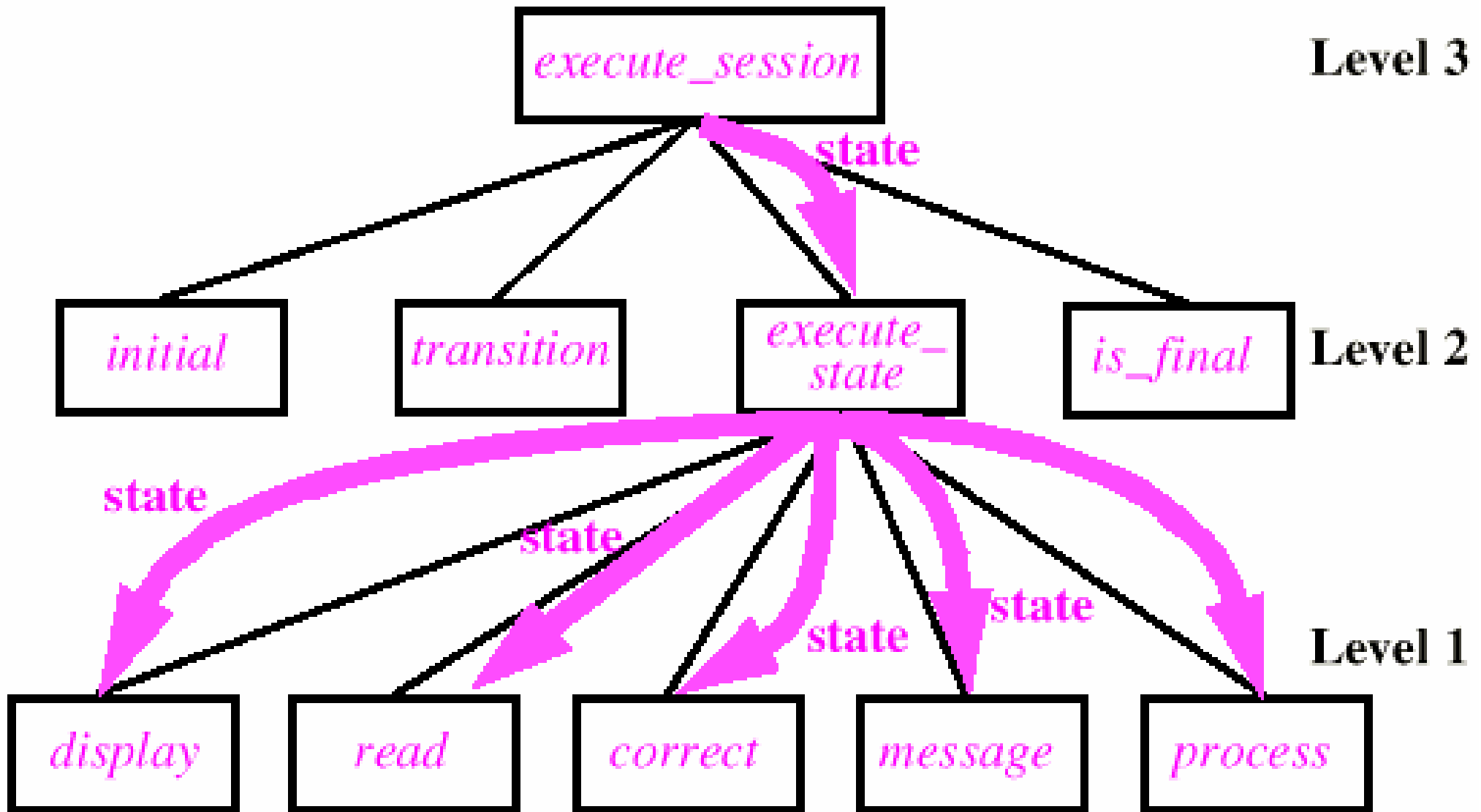
```
/** Execute the actions associated with state s,  
 * returning into c the user's choice for the next state  
 */  
void execute_state (int s, int& c)  
{  
    ANSWER a;  
    bool ok = FALSE;  
  
    while(!ok)  
    {  
        display(s);  
        read(s, a); // a is transferred by reference  
        ok = correct(s, a);  
        if (!ok)  
            message(s, a);  
    }  
    process (s, a);  
    c = next_choice(a);  
}
```



# חתימת המתודות

```
void execute_state (STATE s, CHOICE& c);  
void display (STATE s);  
void read (STATE s, ANSWER& a);  
bool correct (STATE s, ANSWER a);  
void message (STATE s, ANSWER a);  
void process (STATE s, ANSWER a);
```

# זרימת המידע



# נקמת הנתונים

**inspect**

*S*

**when *Initial* then**

...

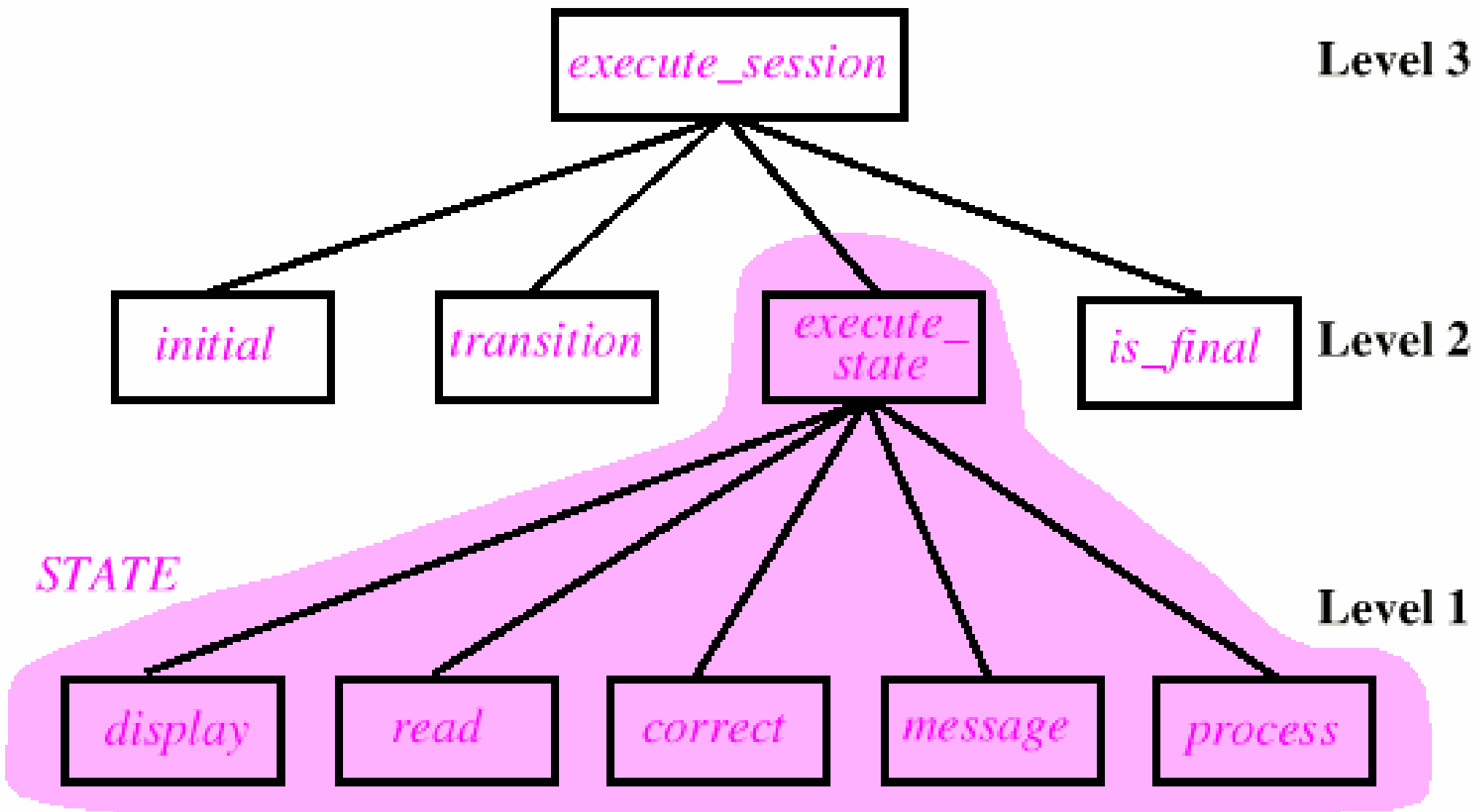
**when *Enquiry\_on\_flights* then**

...

...

**end**

# STATE תכונות





# STATE כמחלקה

```
class STATE
{
public:
    ANSWER input;
    int choice;

    virtual void execute();
    virtual void display();
    virtual void read();
    virtual bool correct();
    virtual void message();
    virtual void process();
};
```

# STATE כמחלקה

```
class STATE
{
    /** User's answer to questions asked in this state */
    ANSWER input;

    /** User's choice for next step */
    int choice;

public:
    /** Display panel associated with current state. */
    virtual void display() = 0 ;

    /** Is input a correct answer? */
    virtual bool correct() = 0;
};
```

# STATE כמחלקה

```
/** Execute actions associated with current state
 * and set choice to denote user's choice for next state.
 * @post: ok
 */
void STATE::execute()
{
    bool ok = FALSE;

    while(!ok)
    {
        display();
        read();
        ok = correct();

        if (!ok)
            message();

        process();
    }
}
```

# STATE כמחלקה

```
class STATE
{
    /** get user's answer into input and choice into choice
     */
    virtual void read() = 0;

    /** Output error message corresponding to input
     * @pre: !corrent()
     */
    virtual void message() = 0;

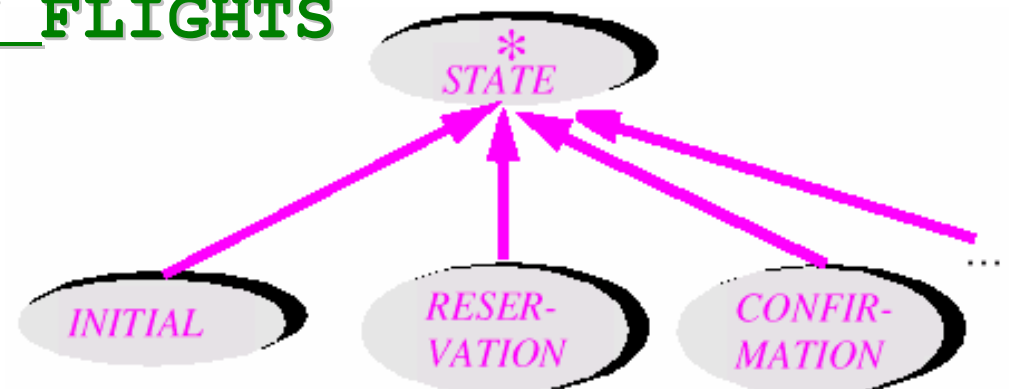
    /** Process input
     * @pre: correct()
     */
    virtual void process() = 0;
};
```

# מצבים ספציפיים יורשים מ STATE

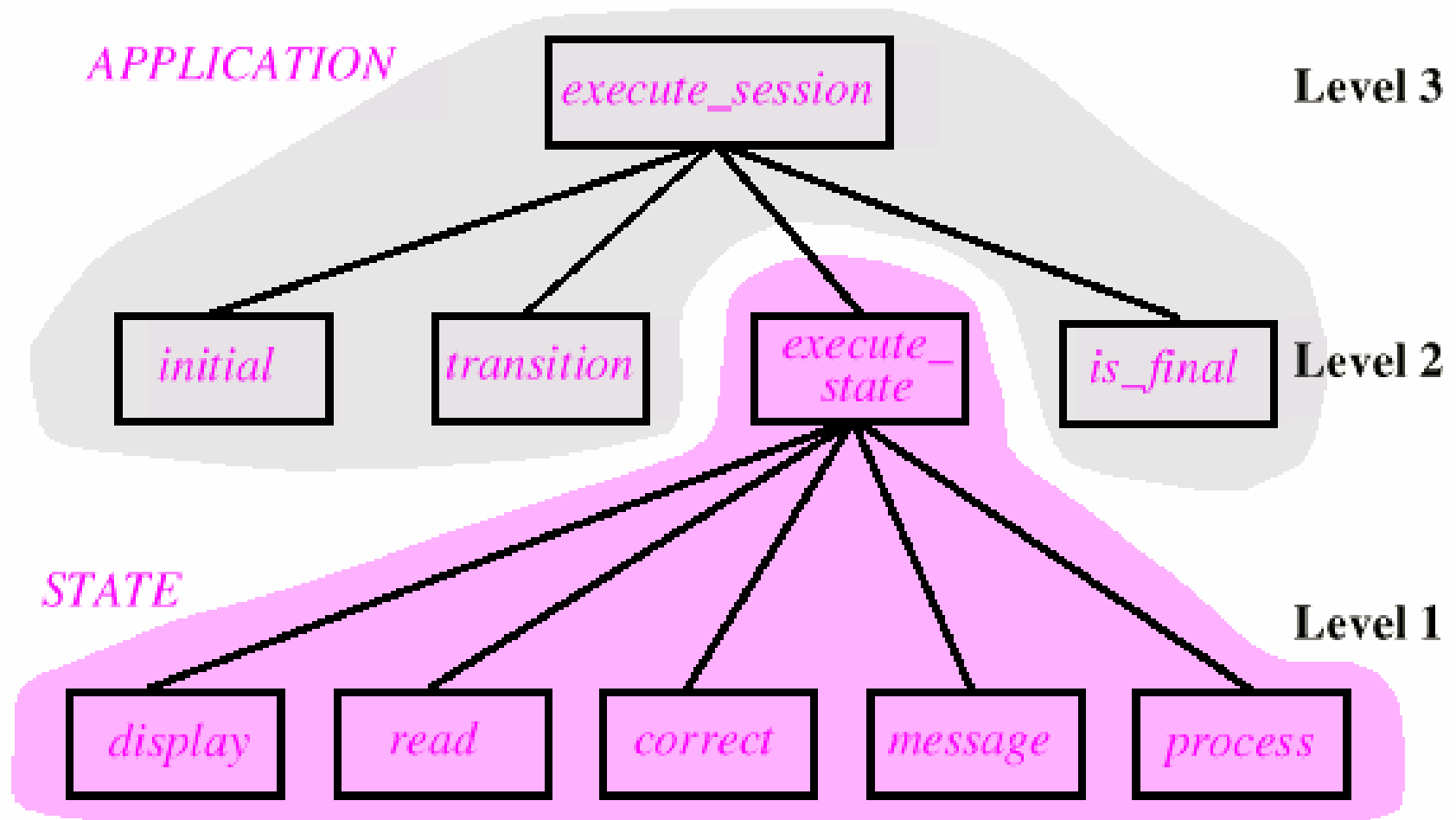
```
class ENQUIRY_ON_FLIGHTS : public STATE
{
    void display()
    { /* Specific display procedure */ }
```

... And similarly for read, correct, message and process...

```
}; // class ENQUIRY_ON_FLIGHTS
```



# תכונות של STATE ושל APPLICATION



# המחלקה APPLICATION

## משתמשת ב STATE

```
/** "Interactive panel-driven applications"
 *  inv: transition.size1() = associated_state.size()
 */
class APPLICATION
{
public:
    /** Allocate application with n states and m
     *  possible choices
     */
    APPLICATION (int n, int m) :
        transition (n+1,m+1), associated_state(n+1)
    {}

    /** Initial state's number */
    const int initial;
```

# המחלקה APPLICATION

```
/** Perform a user session */  
void execute()  
{  
    STATE *st;  
    int st_number;  
  
    st_number = initial;  
    while(st_number != 0)  
    {  
        st = associated_state.item(st_number);  
        st->execute();  
        // This refers to the execute procedure of STATE  
  
        st_number = transition.item(st_number, st->choice);  
    }  
}
```



# המחלקה APPLICATION

```
/** Element change
 * Enter state st with index sn.
 * @pre: 0 < sn && sn <= associated_state.size()
 */
void put_state (STATE st, int sn)
{
    associated_state.put (st, sn);
}

/** Define state number sn as the initial state
 * @pre: 0 < sn && sn <= associated_state.size()
 */
void choose_initial (int sn)
{
    initial = sn;
}
```

# המחלקה APPLICATION

```
/** Enter transition labeled label from state number
 * source to state number target
 * @pre: 1 <= source
 * @pre: source <= associated_state.size()
 *
 * @pre: 0 <= target
 * @pre: target <= associated_state.size()
 *
 * @pre: 1 <= label; label <= transition.size2()
 */
void put_transition (int source, int target, int label)
{
    transition.put(source, label, target);
}
```

# המחלקה APPLICATION

**private:**

```
C2DVector<int> transition;
```

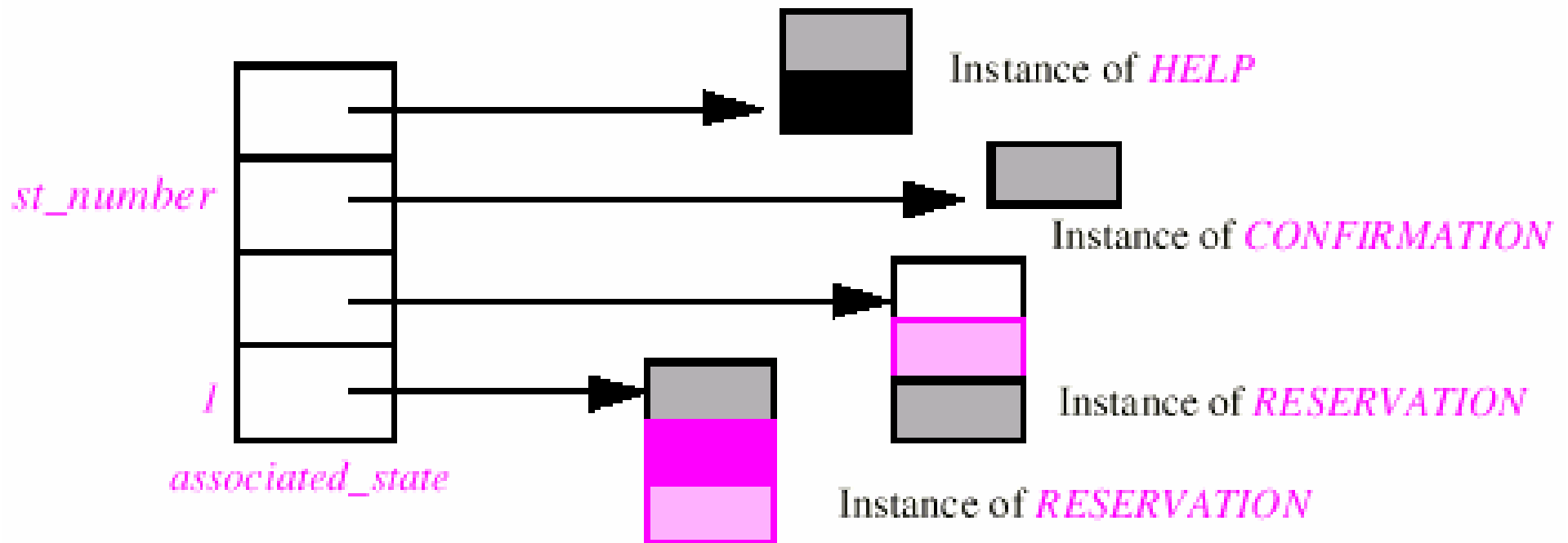
```
vector<STATE *> associated_state;
```

```
//...
```

```
}; // class APPLICATION
```

# המחלקה APPLICATION

■ היישום מחזיק מערך פולימורפי של STATES



# בניית יישום

## ■ אתחול עצם של APPLICATION:

```
APPLICATION air_reservation(number_of_states,  
    number_of_possible_choices);
```

## ■ הגדרים המצבים השונים ומספורם:

```
air_reservation.put_state(s, i);
```

## ■ בחירת המצב ההתחלתי:

```
air_reservation.choose_initial(i0);
```

## ■ הגדרת המעבר ממצב למצב:

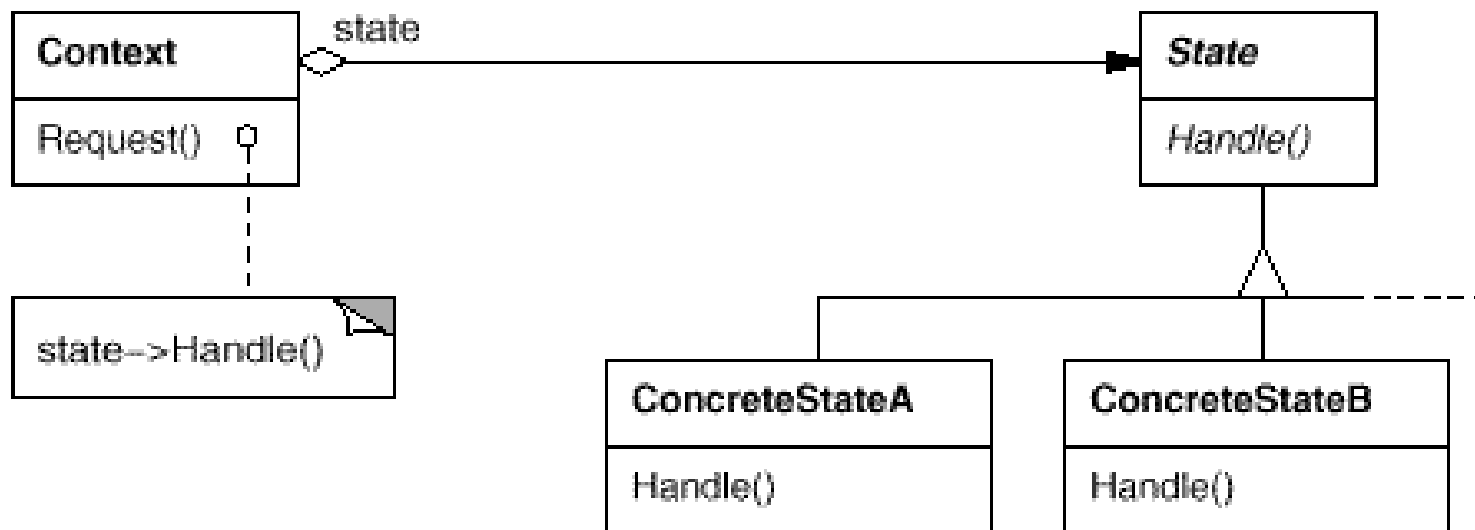
```
air_reservation.enter_transition(sn, tn, l);
```

## ■ הפעלת היישום:

```
air_reservation.execute();
```

# תבנית העיצוב STATE

■ המקרה הכללי:



■ כלפי חוץ נראה כאילו ה context משנה את טיפוסו בצורה דינאמית

# תבנית העיצוב STATE

- התבנית מרכזת התנהגות תלוית מצב למחלקה נפרדת
  - המצבים עצמם לא 'מודעים' לקיומם של מצבים אחרים ולא יושפעו מהוספה/הסרה/שינוי של מצבים אחרים
- המעבר בין המצבים השונים מופיע בקוד בצורה מפורשת
- אם מצבי הנגזרת לא מכילים שדות ניתן להשתמש באותם מצבים עבור כמה יישומים במקביל (כמה TCPConnection למשל)

# חלוקה למחלקות

הערות



# מציאת המחלקות

- במערכות רבות קיימת 'אינפלציה' של מחלקות. להלן מספר מאפיינים למחלקות לא נחוצות:
  - מחלקה שמבצעת פעולה אחת בדר"כ צריכה להיות מתודה של מחלקה אחרת (דלת של מעלית)
  - "המחלקה שלי עושה..." – מחלקה אמיתית לא עושה דבר אלא מציעה שרותים
  - שמות ביצועיים (imperative) – מחלקה לא תיקרא בשם פועל

# מציאת המחלקות

- חלוקה בוסרית (premature classification) – ירושה מבטאת הפשטה של המערכת. כל עוד המערכת לא מסובכת אין צורך לפשט (SAN\_FRANCISCO ו-HOUSTON יורשות מ CITY)
- מחלקות ללא פקודות – בדרך כלל מעידות על struct ולא מחלקה
- מחלקות המבטאות כמה הפשטות

# שם המחלקה

■ שם המחלקה יהיה:

– שם עצם (אולי מתואר) כגון: *TREE, LINKED\_LIST, LINE\_DELETION*

או

– (במקרה שהמחלקה מופשטת) תואר, כגון: *COMPARABLE, NUMERIC*