



תכנות מונחה עצמים בשפת C++

אוהד ברזילי
אוניברסיטת תל אביב



ירושה מרובה

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



דוגמאות מעולם הבעיה

- עוזר הוראה הוא גם סטודנט (תלמיד מחקר) וגם איש סגל (חבר בארגון הסגל הזוטר)
- היחס is-a מתקיים עבור 2 ה'כובעים' של עוזר ההוראה ולכן הוא אמור לרשת ממחלקות שמייצגות את שני התפקידים
- זו אינה בעיה תיאורטית - למתרגל שני כרטיסי קורא בספריה (סטודנט וסגל) ובכל אחד מהם מוענקות לו זכויות השאלה שונות

3

תכנת מונחה עצמים בשפת C++
אוניברסיטת תל אביב



דוגמאות מעולם הבעיה

- מספר ממשי (REAL) הוא גם מספרי (NUMERIC) וגם בן השוואה (COMPARABLE)

```
class NUMERIC {  
    ...  
    NUMERIC operator+(const NUMERIC);  
    NUMERIC operator-(const NUMERIC);  
};  
  
class COMPARABLE {  
    ...  
    bool operator<(const COMPARABLE);  
    bool operator<=(const COMPARABLE);  
};
```

4

תכנת מונחה עצמים בשפת C++
אוניברסיטת תל אביב



דוגמאות מעולם הבעיה

- ולכן אך סביר שמספר ממשי (REAL) גם יירש מהמחלקות NUMERIC ו-COMPARABLE

```
class REAL : public NUMERIC , public COMPARABLE {  
    ...  
};
```



דוגמאות מעולם הבעיה

- חלון של מערכת ההפעלה הוא גם מלבן (לצורך ההצגה הגאומטרית על המסך) וגם איבר ברשימה מקושרת של חלונות (לצורך מעקב אחרי חלונות צאצאים והורים, BACK)

```
class WINDOW : public RECTANGLE , public TREE<WINDOW> {  
    ...  
}
```

- המחלקה TREE מציינת רשימה מקושרת שכל איבר בה הוא רשימה בעצמו

```
class TREE<G> : public list<G> , public CELL<G> {  
    ...  
}
```

- צורה מורכבת (למשל בתוכנה גרפית) היא גם צורה בעצמה וגם רשימה (של צורות)



נישואי נוחות

- האם מחסנית הממומשת בעזרת vector צריכה לרשת ממנו או להכיל אותו?
- VECTORED_STACK תירש מ:
 - STACK את הפונקציונליות (הממשק)
 - vector את המימוש
- מתי ירושה מרובה היא רעיון טוב?
- שפות אחרות אינן מאפשרות ירושה מרובה (smalltalk, JAVA) אלא רק ירושת ממשקים מרובה

7

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב



הבטים של המערכת

- היבט (aspect) של המערכת היא מעין מחלקה המייצגת תכונות רוחביות במערכת התוכנה
- כגון: יעילות, בטיחות, תיעודיות ועוד
- דרך אחת לאכוף תכונות אלו היא ע"י ירושה מרובה:
 - מחלקת הבדיקה של מחלקה X תירש מ TEST ומ-X
 - מחלקה המאפשרת לשמור את מחלקה X בתוך זכרון לא נדיף (persistent) תירש מ STORABLE ומ-X

8

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

התנגשות שמות

- כאשר מחלקה X יורשת ממחלקות A ו-B ולשתיהן יש תכונות (שדות או מתודות) באותו שם – יש לפנות לשדה בשמו המפורש (עם ציון שם המחלקה ע"י ::)
- בשפות אחרות פותרים את הבעיה בדרכים אחרות:
 - ב JAVA אסור לרשת מימוש בירושה מרובה כלל
 - ב Eiffel ניתן לשנות שם של תכונה נורשת

התנגשות שמות - דוגמה

```
class Base1 {
    int m_x;
public:
    Base1(int x) : m_x(x) {}
    void Print () const {
        cout<<"Base1, x="<<m_x<<endl;
    }
};

class Base2 {
    int m_y;
public:
    Base2(int y) : m_y(y) {}
    void Print () const {
        cout<<"Base2, y="<<m_y<<endl;
    }
};
```



התנגשות שמות - דוגמא

```
class Derived : public Base1, public Base2 {
    int m_z;
public:
    Derived(int x, int y, int z)
        : Base1(x), Base2(y) , m_z(z) {}
    void Print() const {
        Base1::Print();
        Base2::Print();
        cout<<"Derived, z="<<m_z<<endl;
    }
};

int main() {
    Derived D(1,2,3);
    D.Print();
    D.Base1::Print();
}
```



סוגי התנגשויות שמות

- o למתודה print אותה המשמעות בשתי מחלקות הבסיס ואולם המימוש שלה שונה
- o במימוש משחק וידאו של המערב הפרוע אנו עשויים לעסוק במחלקות שונות עם מתודות באותו שם, אבל עם משמעות שונה:

```
class Window {
    // ...
    virtual void draw(); // display image
};
class Cowboy {
    // ...
    virtual void draw(); // pull gun from holster
};
class Cowboy_window : public Cowboy , public Window
{
    // ...
};
```



סוגי התנגשויות שמות

- בעוד שבמקרה של print ברור שההגדרה במחלקה הנגזרת מחליפה את שתי הגרסאות בהורים, בדוגמת הבוקר נדרשת זהירות
- נראה פתרון כללי של הבעיה בעזרת טכניקה שנקראת מחלקת ממשק (interface class) – הוספת מחלקת ביניים בתהליך הירושה כדי 'לתקן' את הממשק של המחלקה הנורשת



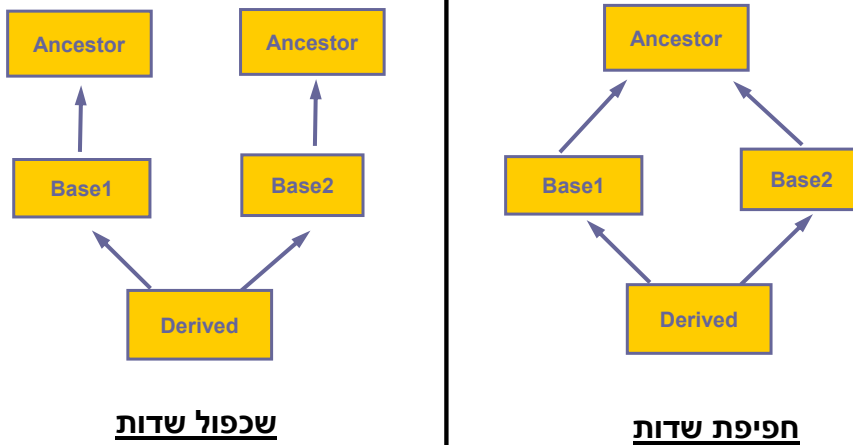
סוגי התנגשויות שמות

```
// interface to Cowboy renaming draw()
class CCowboy : public Cowboy {
public:
    virtual int cow_draw() = 0 ;
    void draw() { cow_draw(); } // override Cowboy::draw()
};

// interface to Window renaming draw()
class WWindow : public Window {
public:
    virtual int win_draw() = 0 ;
    void draw() { win_draw(); } // override Window::draw()
};

class Cowboy_window : public CCowboy , public WWindow {
    // ...
    void cow_draw();
    void win_draw();
};
```

ירושה חוזרת (אב קדמון משותף)



15

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

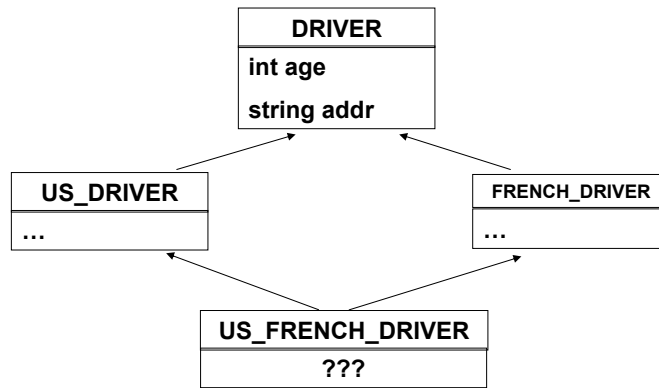
ירושה חוזרת (אב קדמון משותף)

- במקרה כזה תכונות שנורשו פעמיים (דרך שתי מחלקות הביניים) יופיעו פעמיים במחלקה הנגזרת
- זו אינה תמיד ההתנהגות הרצויה
- בשפות מונחות עצמים ניתן להגדיר האם אנו רוצים שכפול של שדות אלו או חפיפה שלהם

16

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

שכפול וחפיפה של שדות



17

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

שכפול וחפיפה של שדות

- כנראה שלא נרצה ששדה הגיל יופיע פעמיים במחלקה US_FRENCH_DRIVER שני השדות מתייחסים לאותו נתון עצמו ותהיה בעיה (טכנית וקונספטואלית) לתחזק את שניהם

- שדה הכתובת שנורש מ-FRENCH_DRIVER מייצג את כתובתו של הנהג בצרפת ואילו שדה הכתובת שנורש מ-US_DRIVER מייצג את הכתובת בארה"ב. כאן, יש משמעות להכלת שתי כתובות, ובעיית העבודה עם שני השדות היא טכנית בלבד

- על מימוש שתי האסטרטגיות השונות ב C++ נרחיב בתרגול

18

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב



ירושה מרובה ב Eiffel

- בשפת Eiffel מתגברים על בעיית השמות בירושה ובירושה מרובה ע"י הוספת כלים של אי-הגדרה (undefine), שינוי שם (rename) וברירת שדות (select)
- עבור כל מחלקה נורשת מציין המתכנת במפורש אילו תכונות הוא מעוניין לרשת, באיזה שם ומאיזה הורה



ירושה עם תבניות

- כאשר A היא תבנית עם טיפוס פורמאלי T יש משמעות לירושה מרובה מאותה תבנית עצמה עם טיפוסים אקטואלים שונים:

```
class B : public A<int> , public A<float>
{
    //...
}
```

- B שבדוגמא איננה תבנית בעצמה