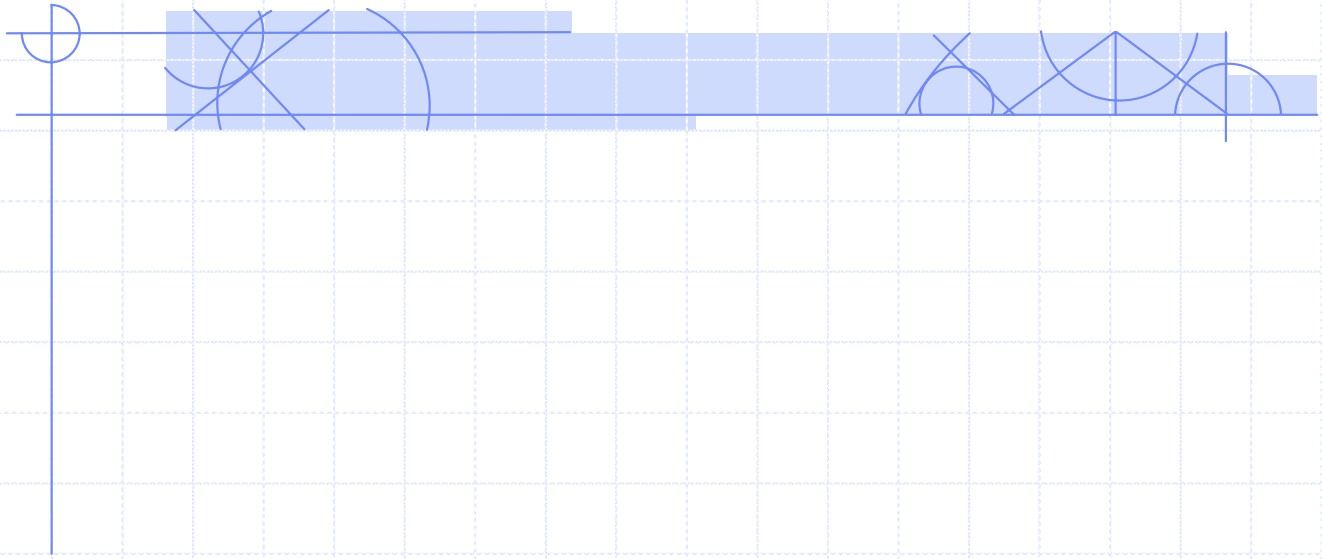
תכנות מונחה עצמים בשפת C++

נושאים מתקדמים בירושה ב C++

אוהד ברזילי
אוניברסיטת תל אביב

טיפוסי זמן ריצה



טיפוס זמן ריצה (RTTI)

- כל עצם יודע בזמן ריצה את טיפוסו (הדינאמי)
- ניתן לקבל מידע זה ע"י שימוש באופרטור typeid() אשר "מוגדר" כך:

```
class type_info;

// pseudo declaration
const type_info& typeid(type_name) throw(bad_typeid);

// pseudo declaration
const type_info& typeid(expression);
```

טיפוס זמן ריצה

- את הטיפוס type_info ניתן להשוות, למיין (ללא קשר להיררכיית הירושה) ולהדפיס
- נדפיס את טיפוסו של ***p*** כך:

```
#include<typeinfo>

void g(Component* p)
{
    cout << typeid(*p).name();
}
```

טיפוס זמן ריצה

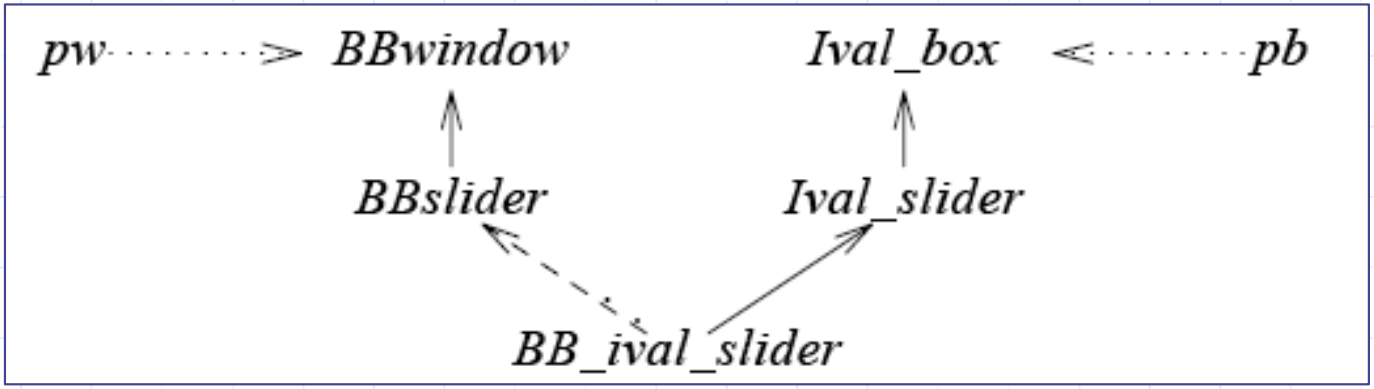
- שימוש במנגנון טיפוסי זמן-ריצה **אינו** תחליף לפולימורפיזם:

```
// misuse of runtime type information:
void rotate(const Shape& r)
{
   if (typeid(r) == typeid(Circle)){
       // do nothing
   }
   else if (typeid(r) == typeid(Triangle)) {
       // rotate triangle
   }
   else if (typeid(r) == typeid(Square)) {
       // rotate square
   }
   // ...
}
```

dynamic_cast<ptr>

- בדר"כ נשתמש בטיפוס זמן ריצה כאשר אנו מקבלים מפונקציה שחתימתה אינה בשליטתנו (callback function) מצביע לטיפוס בעל ממשק סטנדרטי, ואנו רוצים לוודא שהעצם המוצבע הוא מחלקה "שלנו" המממשת ממשק זה
- את בדיקת הטיפוס נעשה תוך כדי המרת טיפוס ע"י האופרטור dynamic_cast
- אם המרת הטיפוס נכשלת מוחזר NULL
- אם מחלקת הבסיס אינה פולימורפית (אינה מכילה מתודות virtual) המרת הטיפוס נכשלת

dynamic_cast<ptr>

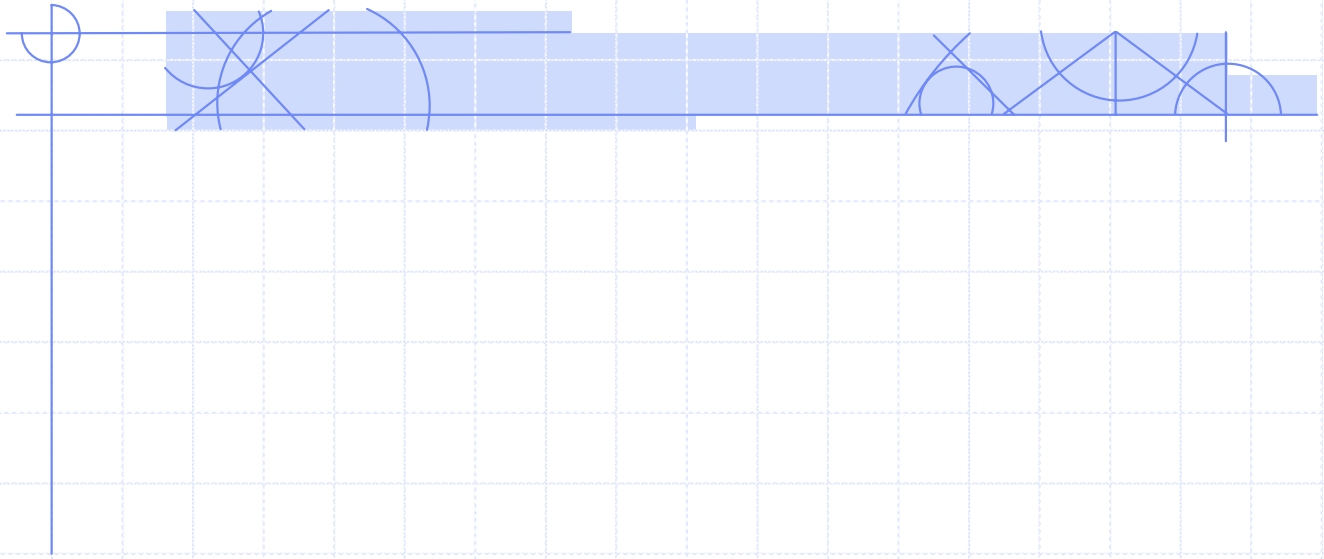



```
void  my_event_handler (BBwindow* pw)
{
        if (Ival_box* pb = dynamic_cast<Ival_box*> (pw) )      // does pw point to an Ival_box?
                pb->do_something ( ) ;
        else {
                // Oops! unexpected event
        }
}
```

סוגי המרות

- המרה מבסיס לנגזרת נקראת downcast
- המרה לנגזרת לבסיס נקראת upcast
- המרה בין "אחים" נקראת crosscast
(lval-box ל- BBwindow)

הרשאות גישה וירושה



הכל נשאר במשפחה

- תכונות שהוגדרו במחלקה מסוימת כפרטיות (private) נגישות רק למחלקה עצמה ולחבריה (מחלקות ומתודות שהוגדרו friend)
- מחלקה הנורשת ממחלקה זו אינה נבדלת משאר העולם – גם היא אינה יכולה לגשת לשדות ה private של הוריה
- לדעת חוגים מסוימים בעולם התוכנה הדבר עומד בסתירה להתייחסות ליחס הירושה כיחס is-a

הכל נשאר במשפחה

- כדי לאפשר בכל זאת יחס מועדף למחלקות יורשות הוגדרה רמת גישה נוספת של "תכונות מוגנות" (protected)
- תכונות שהוגדרו ככאלה זמינות למחלקה שהגדירה אותן ולמחלקות היורשות ממנה אך אינן זמינות ל"שאר העולם"

	Base	Derived	Other
private	☑	☒	☒
protected	☑	☑	☒
public	☑	☑	☑

private vs. protected

- הפילוסופיה של C++ טוענת שייצוג (שדות) צריך להיות private
- ממש כשם שהספק מונע מהציבור הרחב להתעסק במימוש ע"י אספקת ממשק מוגדר היטב של מתודות ציבוריות כך עליו לנהוג גם עבור צאצאיו
- הנימוק הוא שהספק לא מכיר את צאצאיו, אין לו שליטה עליהם ואין לו דרך לוודא שהם עושים שימוש נאות בכוח שניתן להם
- בהקשר זה נציין כי מחלקה נגזרת יכולה להעביר את עצמה לכל פונקציה הדורשת את הוריה ובכך להוציא "שם רע" להורים

הקשחת הסיווג

- מחלקה רשאית **להחמיר** את הרשאות הגישה של המחלקה ממנה היא יורשת
- הדבר נעשה ע"י הכרזה על **הרשאת מינימום** בשורת הגדרת המחלקה
- ניתן **להקל** את הרשאת הגישה של תכונות מסוימות (אך לא של תכונות שאינן זמינות בנגזרת) ע"י הכרזה עליהן תחת ההרשאה המתאימה (עם או בלי המילה using)

סיכום הרשאות גישה

Base		Derived1 : ____ Base			Derived2 : public Derived1			Derived1 in Main		
		Pub	**Prt**	**Prv**	**Pub**	**Prt**	**Prv**	**Pub**	**Prt**	**Prv**
Private	☑	☒	☒	☒	☒	☒	☒	☒	☒	☒
Protected	☑	☑ **(Prt)**	☑ **(Prt)**	☑ **(Prv)**	☑ **(Prt)**	☑ **(Prt)**	☒	☒	☒	☒
Public	☑	☑ **(Pub)**	☑ **(Prt)**	☑ **(Prv)**	☑ **(Pub)**	☑ **(Prt)**	☒	☑	☒	☒

סוגי ירושה

- ירושת public מכונה גם subtyping , יחס is-a
- ירושת private ו protected מכונות גם ירושות מימוש
- בירושה עם הרשאה X אומרים שלגזרת יש בסיס עם הרשאה X
- ירושות מימוש משמשות כלי נוח לביצוע האצלה

דוגמא: ירושה עם הרשאות

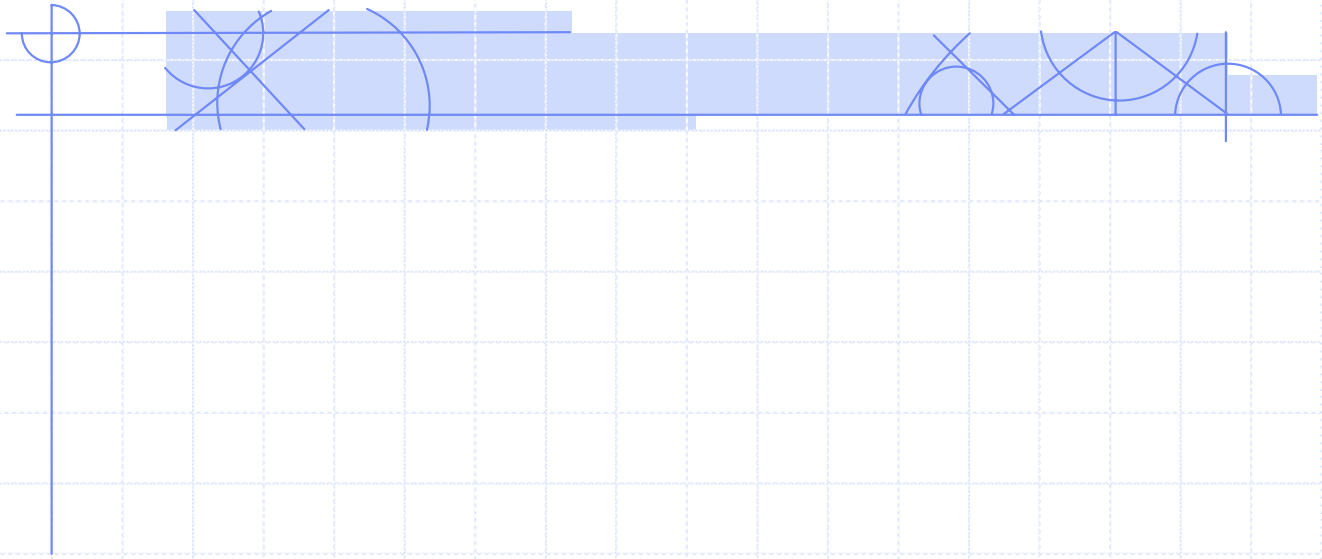
```
class Base
{
   int m_prv;

protected:
   int m_prt;

public:
   int m_pub1;
   int m_pub2;
   Base(int prv, int prt, int pub) : m_prv(prt),
        m_prt(prt), m_pub1(pub), m_pub2(pub) {}
   void func1() {}
   void func2() {}
};
```

```
class Derived1 : private Base
{
   int i;

protected:
   int j;

public:
   Base::func1;
   Base::m_pub1;
   Base::m_prt;

   int k;

   Derived1():
        Base(1,2,3) , i(0), j(0), k(0) {}

   void Do()
   {
        // m_prv = 23; // no aceess - private in Base
        m_prt= 3;      // OK OK - protect in Base
        func2();
   }
};
```

```
int main()
{
   Derived1 D1;

   // D1.i = 0; // no aceess - private in Derived1
   // D1.j = 0; // no aceess - protected in Derived1
   D1.k = 0;    // OK

   // D1.m_prv = 0;  // no aceess - private in Base
   D1.m_prt = 0;     // OK - became public in Derived1
   D1.m_pub1 = 0;    // OK - restored to public
   // D1.m_pub2 = 0; // no aceess - became private in Derived1

   D1.func1();    // OK - restored to public
   // D1.func2(); // no aceess - became private in Derived1
}
```

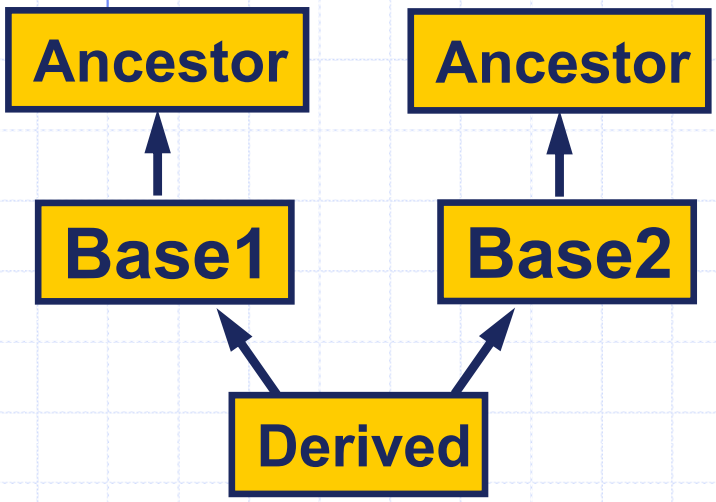
```
class Derived2 : public Derived1
{
public:
   void Do()
   {
        Derived1::Do();
        // m_prt = 0;// no access - became private
        // i = 0;    // no aceess - private in Derived1
        j = 0;       // OK - Protected in Derived1
        func1();     // OK - retored to public
        // func2();  // no access - became private
   }
};

int main()
{
   Derived2 D2;

   D2.Do();
   D2.Derived1::Do();
}
```

ירושה מרובה



אב קדמון משותף

- שדות המוכלים באב קדמון משותף יופיעו פעמיים בצאצא (הנכד)
- מאיפה מגיעים העותקים הכפולים?



- **Ancestor A;**

A:

Ancestor (members)

- **Base1 B1; Base2 B2;**

B1:

Ancestor (members)
Base1 (additional)

B2:

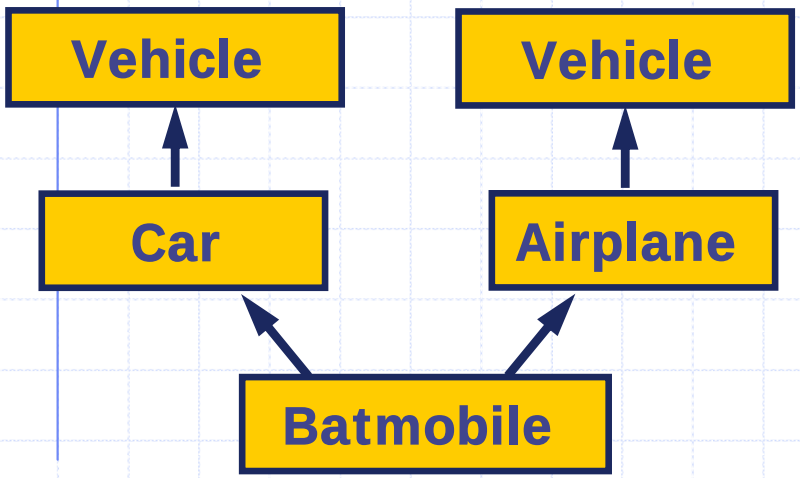
Ancestor (members)
Base2 (additional)

- **Derived D;**

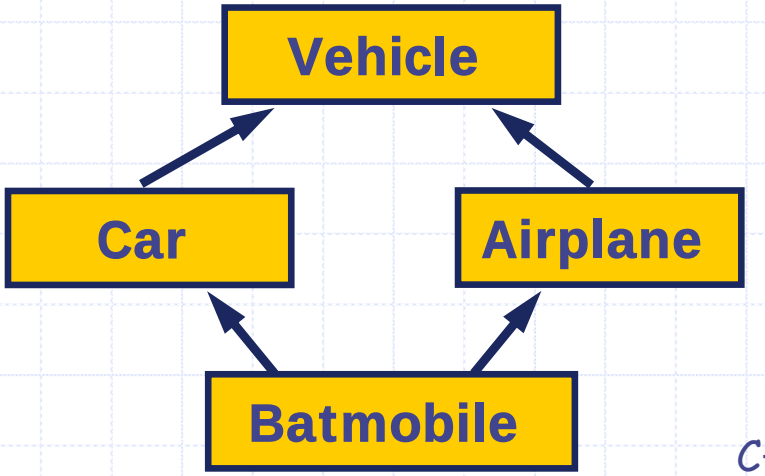
D:

Ancestor (members)	Base1
Base1 (additional)	
Ancestor (members)	Base2
Base2 (additional)	
Derived (additional)	

אב קדמון משותף



או



- C++ מאפשרת למתכנת ברירה בין שתי אפשרויות:
  - ירושה "רגילה": כל מחלקה מכילה את הוריה (כולל שכפול תכונות)
  - ביטול כפילויות: עבור תכונות שהגיעו ב"מסלולים שונים" אל הנכד – ישמר רק עותק אחד
- הבחירה בין שתי האפשרויות תלויה במחלקה הממומשת

copyrights © Elhanan Borenstein

אב קדמון משותף

```
class Vehicle
{
   int m_speed; // in kp"h
   bool m_wings;

public:
   Vehicle(int speed, bool wings = false)
        : m_speed(speed), m_wings(wings) {}

   int Speed () { return m_speed; }

   bool HasWings() { return m_wings; }

   void Print () const
   {
     cout<<"Wee, I have "<< (m_wings?"":"no ")<<"wings"<<endl;
     cout<<"Speed="<<m_speed<<endl;
   }
};
```


copyrights © Elhanan Borenstein

אב קדמון משותף

```
class Car : public Vehicle
{
    bool m_automatic;
public:
    Car(bool automatic) :
          Vehicle(90, false), m_automatic(automatic) {}

    IsAutomatic() { return m_automatic;}
};


class Airplane : public Vehicle
{
    int m_missiles;
public:
    Airplane(int missiles) : Vehicle(1200, true), m_missiles(missiles) {}
    NumMissiles() { return m_missiles; }
};
```


copyrights © Elhanan Borenstein

אב קדמון משותף

```
class Batmobile : public Car, public Airplane
{
public:
   Batmobile(bool automatic, int missiles)
        : Car(automatic), Airplane(missiles) {}
};
```

```
int main()
{
  Batmobile B(true, 6);
  // B.Print();// Ambiguous

  // Now I'm a car
  B.Car::Print();
  // Now I'm an airplane
  B.Airplane::Print();
}
```

Members inherited from Vehicle (twice):	int m_speed bool m_wings
Additional members inherited from Car:	bool m_automatic
Additional members inherited from Airplane:	int m_missiles
Additional members in Batmobile:	NONE

x2

copyrights © Elhanan Borenstein

ירושה וירטואלית

- כאשר ירושה מוגדרת וירטואלית הקומפיילר בונה מחלקה נגזרת כך שתכיל עותק אחד בדיוק של כל השדות שהיגיעו במסלולי הירושה השונים
- הירושה הוירטואלית מוגדרת על מחלקות הביניים ולא על הנכד

אב קדמון משותף – ירושה וירטואלית

Simple Inheritance

- **Ancestor A;**

  A: **Ancestor** (members)

- **Base1 B1; Base2 B2;**

  B1: **Ancestor** (members) **Base1** (additional)

  B2: **Ancestor** (members) **Base2** (additional)

- **Derived D;**

  D: **Ancestor** (members) | **Base1** (additional) — Base1 | **Ancestor** (members) | **Base2** (additional) — Base2 | **Derived** (additional)

Virtual Inheritance

- **Ancestor A;**

  A: **Ancestor** (members)

- **Base1 B1; Base2 B2;**

  B1: near ptr, **Base1** (additional)

  B2: near ptr, **Base2** (additional)

- **Derived D;**

  D: near ptr | **Base1** (additional) | near ptr | **Base2** (additional) | **Derived** (additional)

  Ancestor (members)

  Ancestor (members)

  Ancestor (members)

ירושה וירטואלית

```
class Ancestor
{
public:
   int m_a;
};
```

```
class Derived : public Base1, public Base2
{
public:
   int m_d;
};
```

```
class Base1 : virtual public Ancestor
{
public:
   int m_b1;
};

class Base2 : virtual public Ancestor
{
public:
   int m_b2;
};
```

ירושה וירטואלית

```
int main()
{
   Derived D;

   D.m_a = 10; // OK - No ambiguity
   cout<<D.Base1::m_a<<endl; // Prints 10
   cout<<D.Base2::m_a<<endl; // Prints 10

   D.Base1::m_a = 7;
   cout<<D.Base1::m_a<<endl; // Prints 7
   cout<<D.Base2::m_a<<endl; // Prints 7
}
```

סבים ונכדים

- אחריות המתכנת לוודא שאין למהדר התלבטויות (ambiguities) – למשל אם שתי מחלקות הביניים הסתירו מתודה שנורשה מאב משותף או אם שתי מחלקות הביניים מכילות ערכים שונים לאותו שדה

- בדוגמא הבאה הנכד מאתחל את הבנאי של סבא שלו, הדבר הכרחי כדי לפתור בעיות ambiguities למהדר (ובדר"כ נובע מתכנון לקוי של הירושה)

עוד דוגמא: ירושה וירטואלית

```
class Employee
{
   string    Name;
   int       Salary;

public:
   Employee(const string& name, int salary)
       : Name(name), Salary(salary) {}
   virtual ~Employee() {}

   const string& GetName() { return Name;}
   int GetSalary() { return Salary; }
};
```

עוד דוגמא: ירושה וירטואלית

```
class Manager : virtual public Employee
{
  int Level;

public:
  Manager(const Employee& em, int level)
      : Employee(em), Level(level) {};

  int GetLevel() { return Level; }
};
```

עוד דוגמא: ירושה וירטואלית

```
class Secretary : virtual public Employee
{
   int WordsPerMin;

public:
   Secretary(const Employee& em, int words)
       : Employee(em), WordsPerMin(words) {};

   int GetWordsPerMin() { return WordsPerMin; }
};
```

עוד דוגמא: ירושה וירטואלית

```
class ManagerSecretary :
                public Manager, public Secretary
{
public:
   ManagerSecretary (const Manager& ma,
                              const Secretary& se)
        : Employee(ma), Manager(ma), Secretary(se)
   {}
};
```

עוד דוגמא: ירושה וירטואלית

```
int main()
{
   // Before the hi-tech crisis
   Manager BillGates(Employee("Bill Gates", 1000000), 9);
   Secretary Moneypenny(Employee("Moneypenny", 3000), 60);


   // After the hi-tech Crisis
   ManagerSecretary PoorBillGates(BillGates, Moneypenny);
}
```

- איך קוראים ל PoorBillGates ?

כלומר, מה יחזיר PoorBillGates.GetName() ?

virtual

- שימו לב – אין קשר בין:
  - **מתודות וירטואליות** (תכונה מרכזית בתכנות מונחה עצמים ההופכת את המחלקה לפולימורפית)

  ובין
  - **ירושה וירטואלית** (נושא איזוטרי המתמודד עם שכפול שדות מאב קדמון משותף)