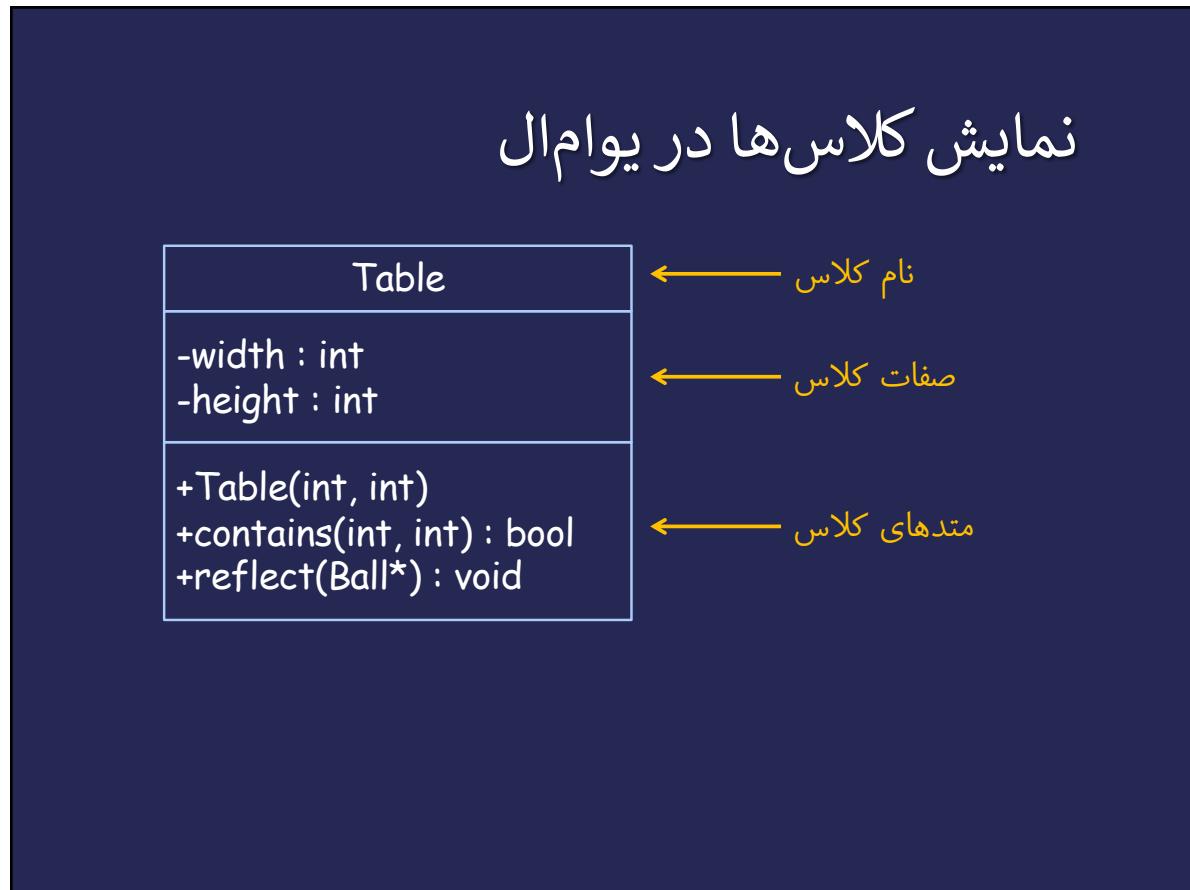


1



2

کلاس‌ها در یوامال و کد برنامه

Table
-width : int
-height : int

+Table(int, int)
+contains(int, int) : bool
+reflect(Ball*) : void

```
class Table {  
public:  
    Table(int, int);  
    bool contains(int, int);  
    void reflect(Ball*);  
private:  
    int width;  
    int height;  
};
```

3

نمایش‌های خلاصه‌تر

Table
-width : int
-height : int

+Table(int, int)
+contains(int, int) : bool
+reflect(Ball*) : void

Table
width
height

Table()
contains()
reflect()

4

نمایش‌های خلاصه‌تر

Table
-width : int
-height : int
+Table(int, int)
+contains(int, int) : bool
+reflect(Ball*) : void

Table
width
height
contains()
reflect()

5

نمایش‌های خلاصه‌تر

Table
-width : int
-height : int
+Table(int, int)
+contains(int, int) : bool
+reflect(Ball*) : void

Table
width
height

6

نمایش‌های خلاصه‌تر

Table
-width : int
-height : int
+Table(int, int)
+contains(int, int) : bool
+reflect(Ball*) : void

Table
contains()
reflect()

7

نمایش‌های خلاصه‌تر

Table
-width : int
-height : int
+Table(int, int)
+contains(int, int) : bool
+reflect(Ball*) : void

Table
contains()

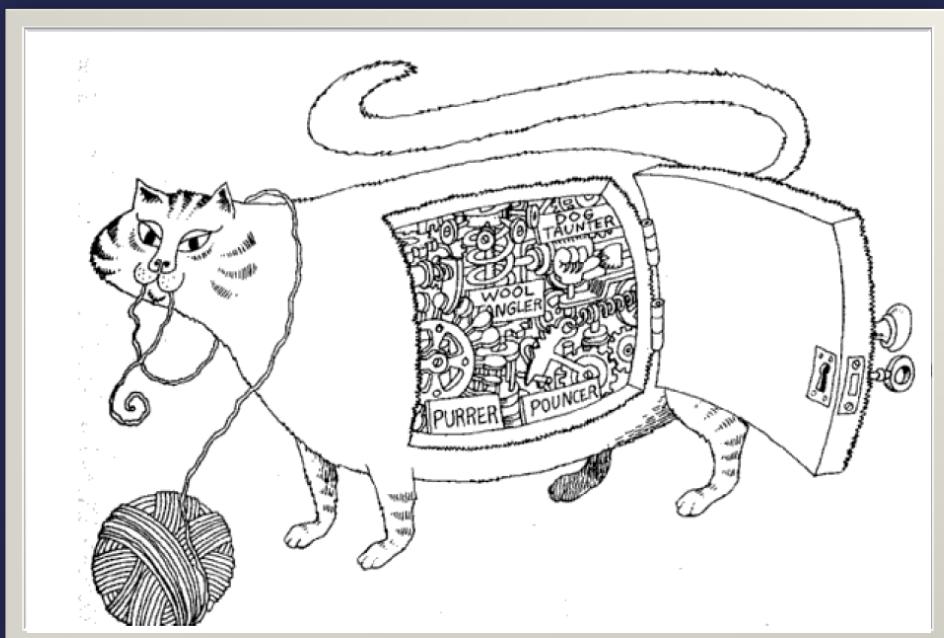
8

نمایش روابط بین کلاس‌ها



```
class Ball {  
    ...  
private:  
    Table* table;  
};
```

9



encapsulation

لafafبندی

10

```
class Table {  
public:  
    Table(int, int);  
    bool contains(int, int);  
    void reflect(Ball*);  
private:  
    int width;  
    int height;  
};
```

کنار هم قرار گرفتن داده‌ها و عملیات به عنوان موجودیتی واحد

11

```
class Table {  
public:  
    Table(int, int);  
    bool contains(int, int);  
    void reflect(Ball*);  
private:  
    int width;  
    int height;  
};  
  
Table::Table(int w, int h) {  
    ...  
}  
  
// body of other methods
```

جداسازی واسط (interface) از
پیاده‌سازی (implementation)

12

```

class Table {
public:
    Table(int, int);
    bool contains(int, int);
    void reflect(Ball*);
private:
    int width;
    int height;
};

Table::Table(int w, int h) {
    ...
}

// body of other methods

```

واسط کلاس Table

واسط یک کلاس مانند
قراردادی است که طبق آن
می‌توان از آن کلاس استفاده
کرد.

قرارداد = protocol

13

```

class Table {
public:
    Table(int, int);
    bool contains(int, int);
    void reflect(Ball*);
private:
    int width;
    int height;
};

Table::Table(int w, int h) {
    ...
}

// body of other methods

```

پیاده‌سازی یک کلاس برای
کلاس‌های دیگر غیرقابل
دسترسی است.

پیاده‌سازی کلاس Table

14

مخفی‌سازی اطلاعات

یک کلاس باید تمام اطلاعاتی را که برای استفاده از آن لازم است به بیرون عرضه کند؛ و نه بیشتر!

15

```
class Table {  
public:  
    Table(int, int);  
    int width;  
    int height;  
};
```

میزی مستطیل که ابعاد آن در دو متغیر به نام‌های `width` و `height` نگهداری می‌شوند.

میزی مستطیل که می‌توان ابعاد آن را از آن پرسید.

```
class Table {  
public:  
    Table(int, int);  
    bool contains(int, int);  
private:  
    int width;  
    int height;  
};
```

```
class Table {  
public:  
    Table(int, int);  
    int get_width();  
    int get_height();  
private:  
    int width;  
    int height;  
};
```

میزی که ... ؟

16



inheritance

وراثت

17

سیستم اطلاعاتی دانشگاه

Student
name
national_code
student_id
get_name()
get_national_code()
get_id()

Employee
name
national_code
emp_code
get_name()
get_national_code()
get_code()
calc_salary()

18

سیستم اطلاعاتی دانشگاه

اطلاعات مشترک بین دو کلاس

Student	Employee
<code>name</code>	<code>name</code>
<code>national_code</code>	<code>national_code</code>
<code>student_id</code>	<code>emp_code</code>
<code>get_name()</code>	<code>get_name()</code>
<code>get_national_code()</code>	<code>get_code()</code>
<code>get_id()</code>	<code>calc_salary()</code>

19

سیستم اطلاعاتی دانشگاه

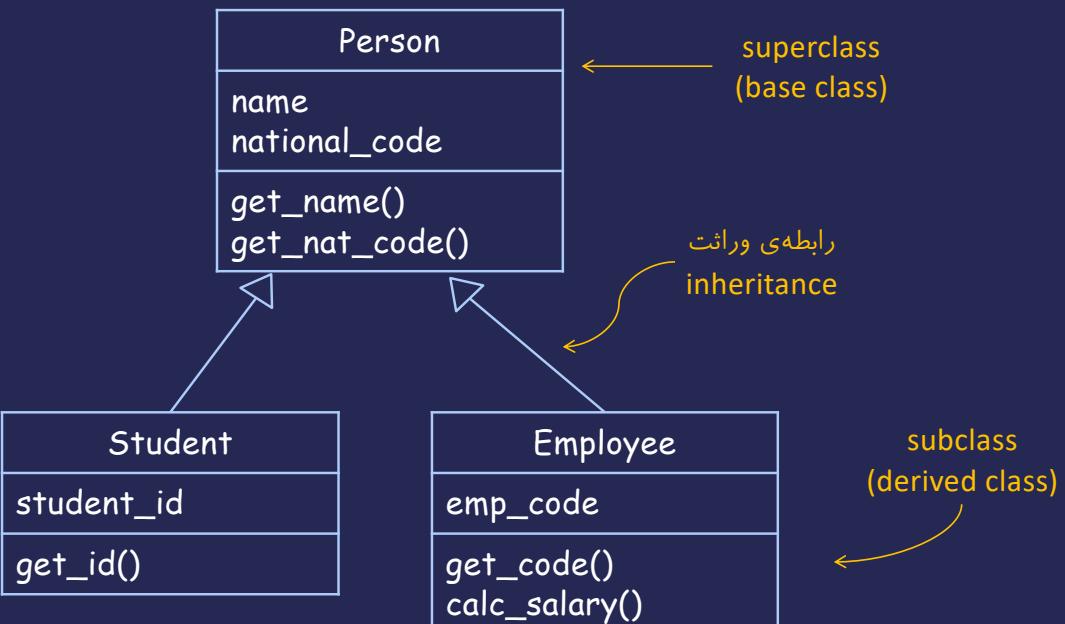
اطلاعات مشترک بین دو کلاس

Student	Employee
<code>name</code>	<code>name</code>
<code>national_code</code>	<code>national_code</code>
<code>student_id</code>	<code>emp_code</code>
<code>get_name()</code>	<code>get_name()</code>
<code>get_national_code()</code>	<code>get_code()</code>
<code>get_id()</code>	<code>calc_salary()</code>

بدنه‌ی یکسان

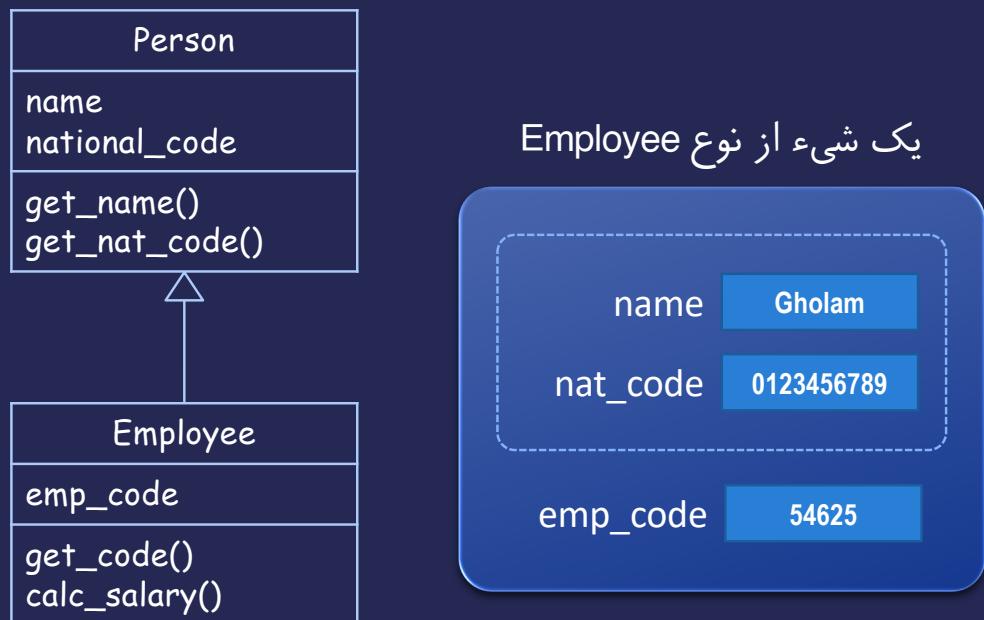
20

فاکتورگیری اطلاعات مشترک



21

اشیاء از نوع زیرکلاس‌ها



22

متن این مثال را ببینید

ramtung / apnotes

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki

Branch: master apnotes / 11_Inheritance / 01_inheritance.cpp

ramtung fixed compile errors and filenames

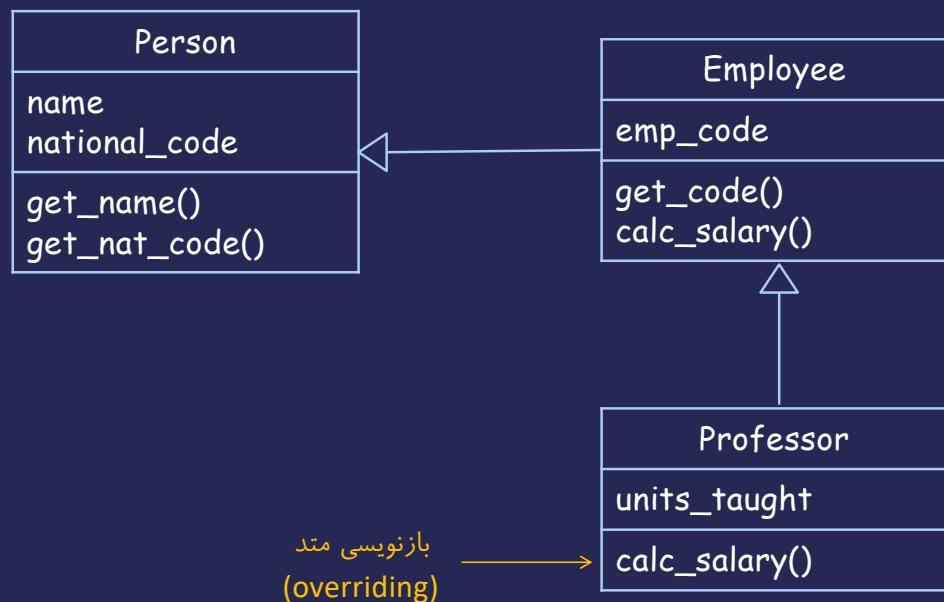
1 contributor

54 lines (45 sloc) | 1.05 KB

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 class Person {
6 public:
7     Person(string n, string c)
8         : name(n), nat_code(c) {}
```

23

اضافه کردن کلاس استاد



24

متن این مثال را ببینید

ramtung / apnotes

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki

Branch: master → apnotes / 11_Inheritance / 02_inheritance.cpp

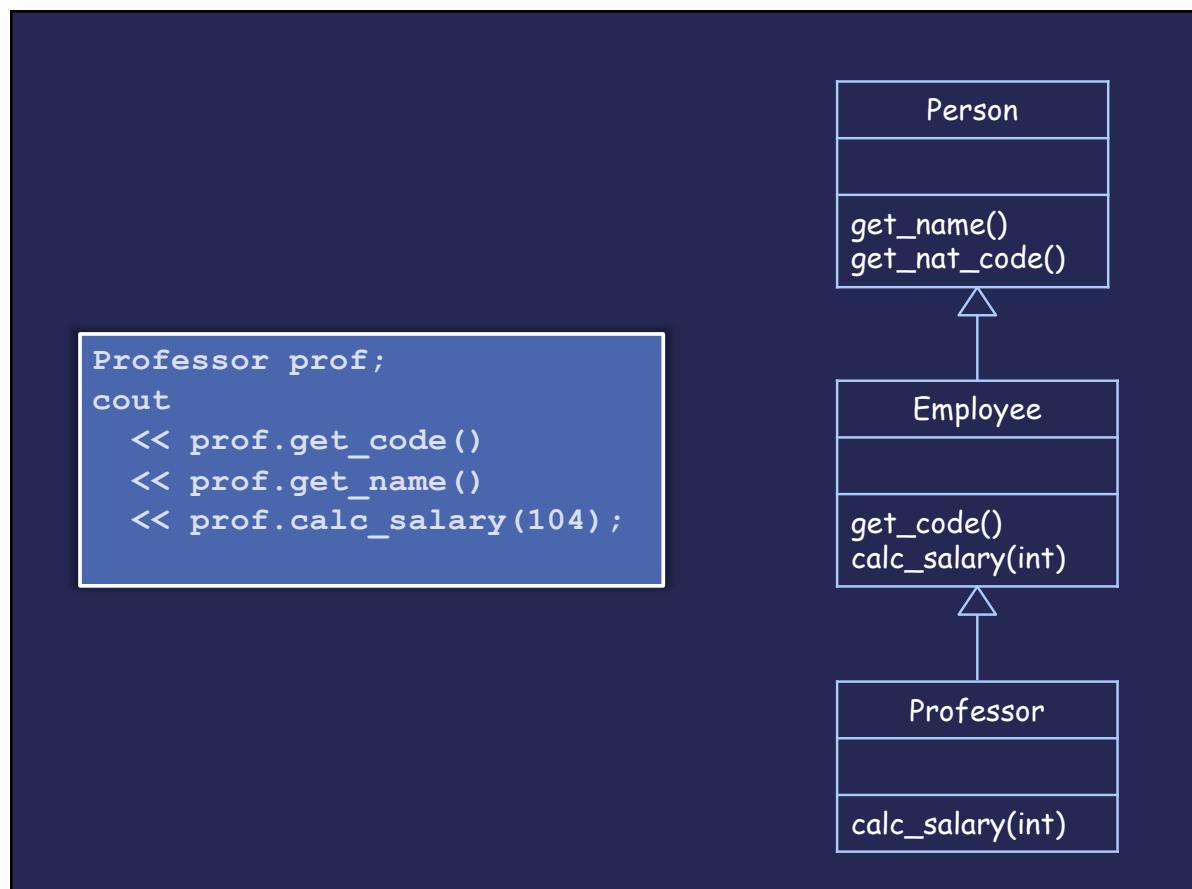
ramtung fixed compile errors and filenames

1 contributor

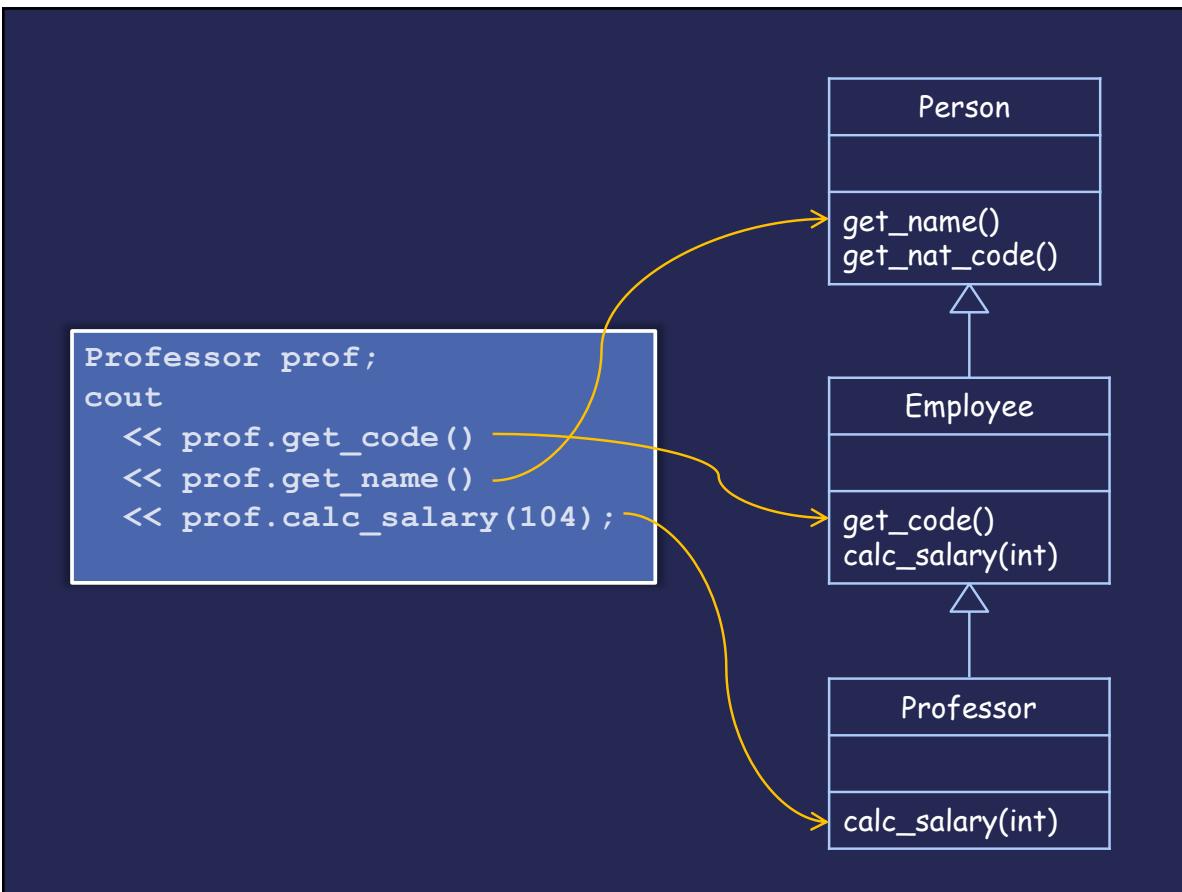
62 lines (51 sloc) | 1.28 KB

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 class Person {
6 public:
7     Person(string n, string c) : name(n), nat_code(c) {}
```

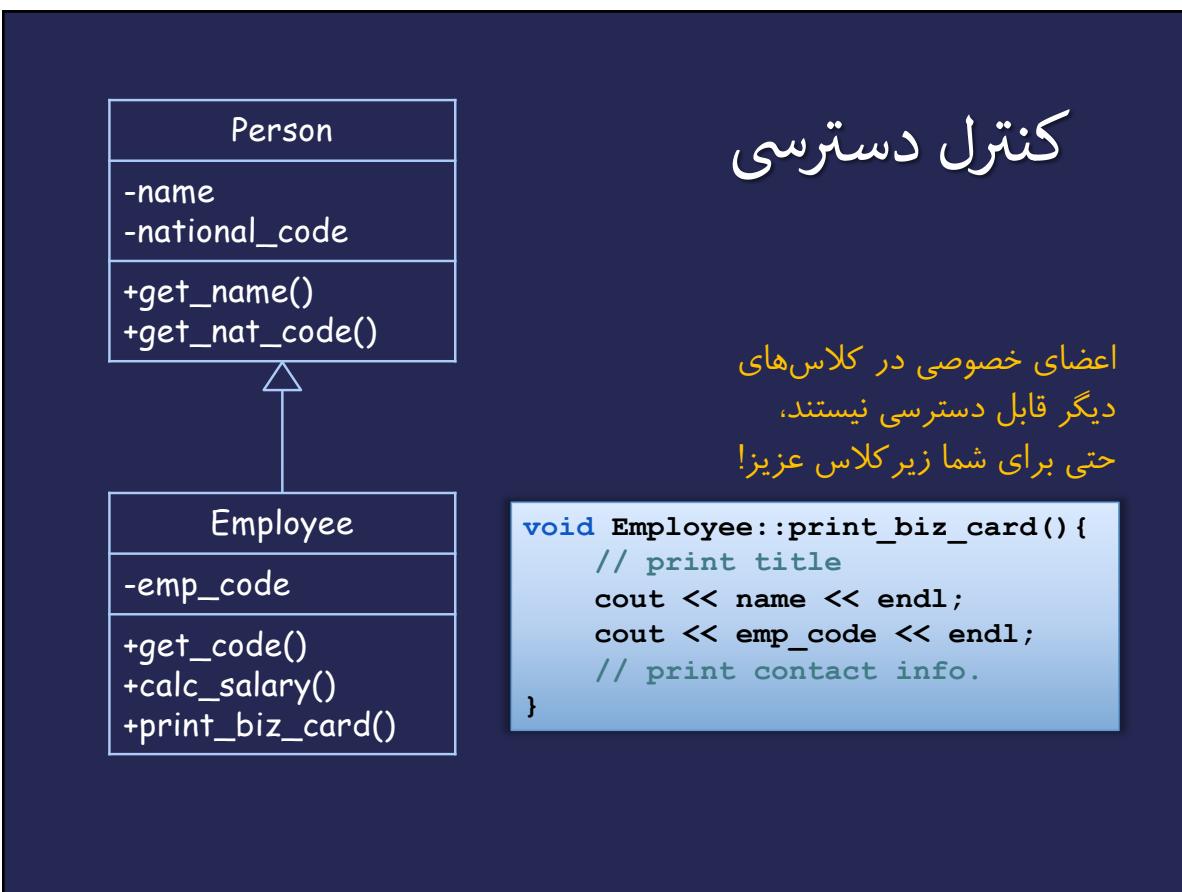
25



26

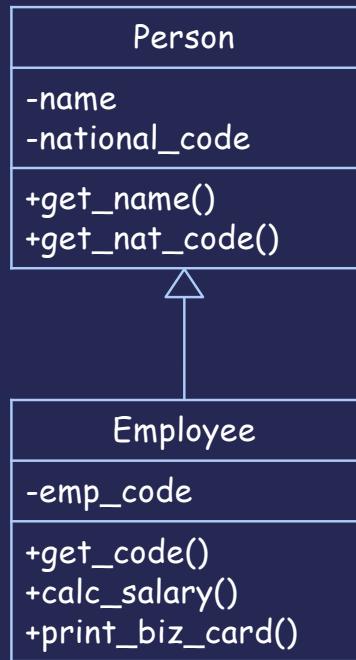


27



28

راه حل اول

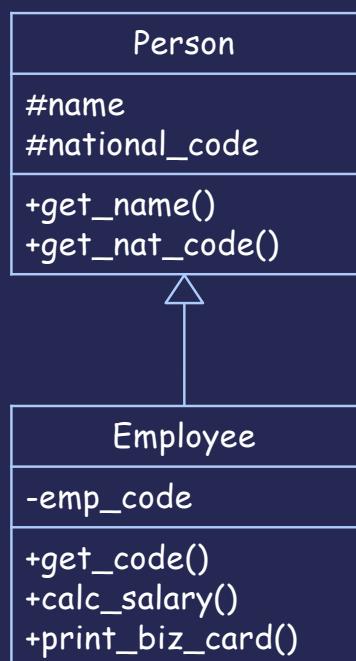


استفاده از واسط عمومی سوپر کلاس

```
void Employee::print_biz_card() {
    // print title
    cout << get_name() << endl;
    cout << emp_code << endl;
    // print contact info.
}
```

29

دسترسی محافظت شده



```
class Person {
protected:
    string name;
    string national_code;
public:
    // as before
};
```

```
void Employee::print_biz_card() {
    // print title
    cout << name << endl;
    cout << emp_code << endl;
    // print contact info.
}
```

30

رابطه‌ی وراثت

- فاکتورگیری کدهای مشترک
- زیرکلاس اعضای ابرکلاس را به ارث می‌برد
- بازنویسی متدهای به ارث رسیده ممکن است
- فیلدرا را نمی‌توان بازنویسی کرد

31

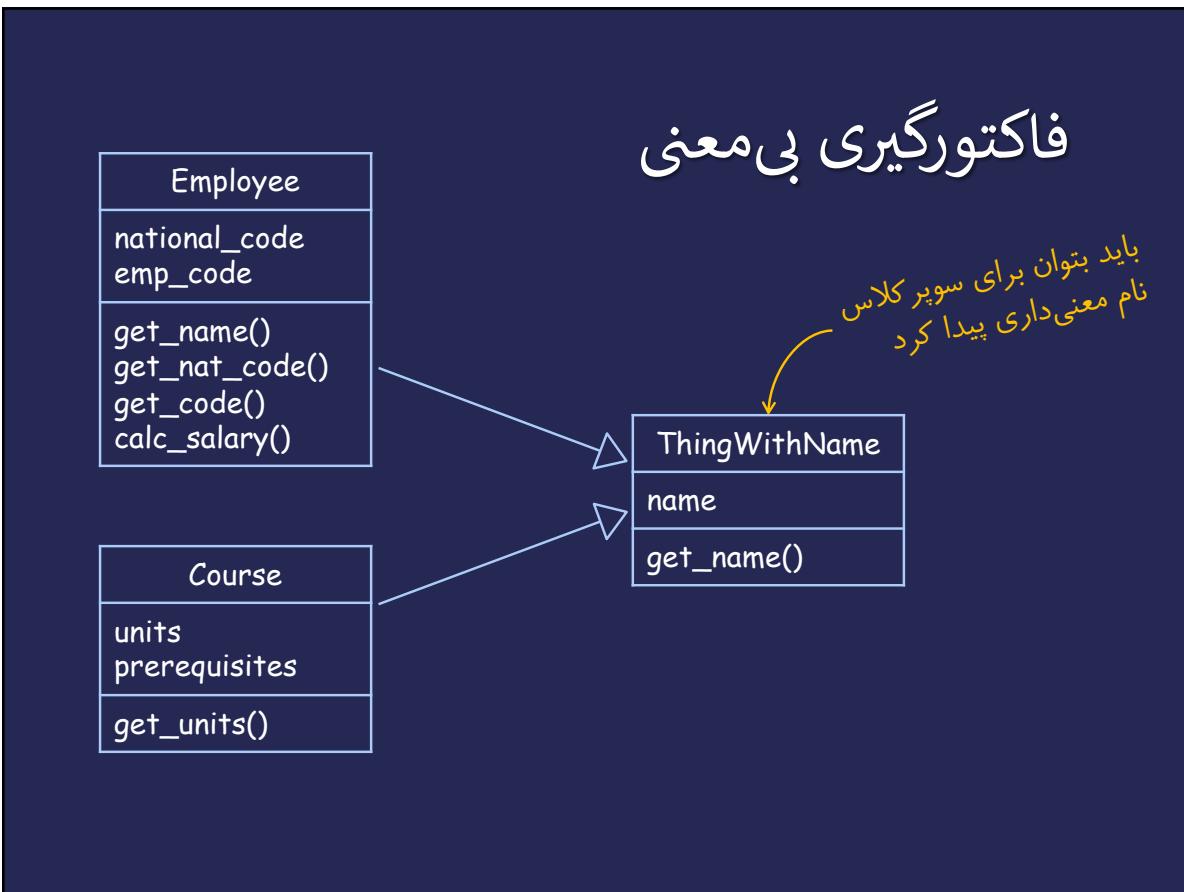
فاکتورگیری بی معنی

Employee
<code>name</code>
<code>national_code</code>
<code>emp_code</code>
<code>get_name()</code>
<code>get_nat_code()</code>
<code>get_code()</code>
<code>calc_salary()</code>

Course
<code>name</code>
<code>units</code>
<code>prerequisites</code>
<code>get_name()</code>
<code>get_units()</code>

32

فاکتورگیری بی معنی



33

استفاده مجدد از کد



34

تاریخ تولد شخص

Person
name
national_code
day
month
year
get_name()
get_national_code()

Date
day
month
year
set_date()
print()
add_one_day()

35

استفاده مجدد نابهجا!

Date
day
month
year
set_date()
print()
add_one_day()

Person
name
national_code
get_name()
get_national_code()

به هدف استفاده مجدد از کد
کلاس تاریخ، با این استدلال
که هر فرد تاریخ تولد دارد!



```
Person gholam();  
gholam.add_one_day();  
gholam.print();
```

36

آزمون IS-A

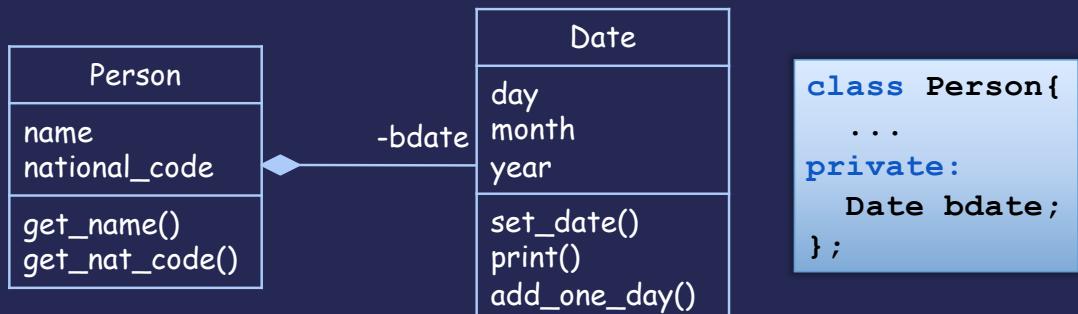
◊ پیش از به کار گیری رابطه‌ی وراثت آن را با آزمون

: IS-A محک بزنید:

- ◊ Student IS-A Person
- ◊ Professor IS-A(n) Employee
- ⇒ Person IS-A Date

37

استفاده‌ی مجدد با رابطه‌ی HAS-A



Person HAS-A Date ✓

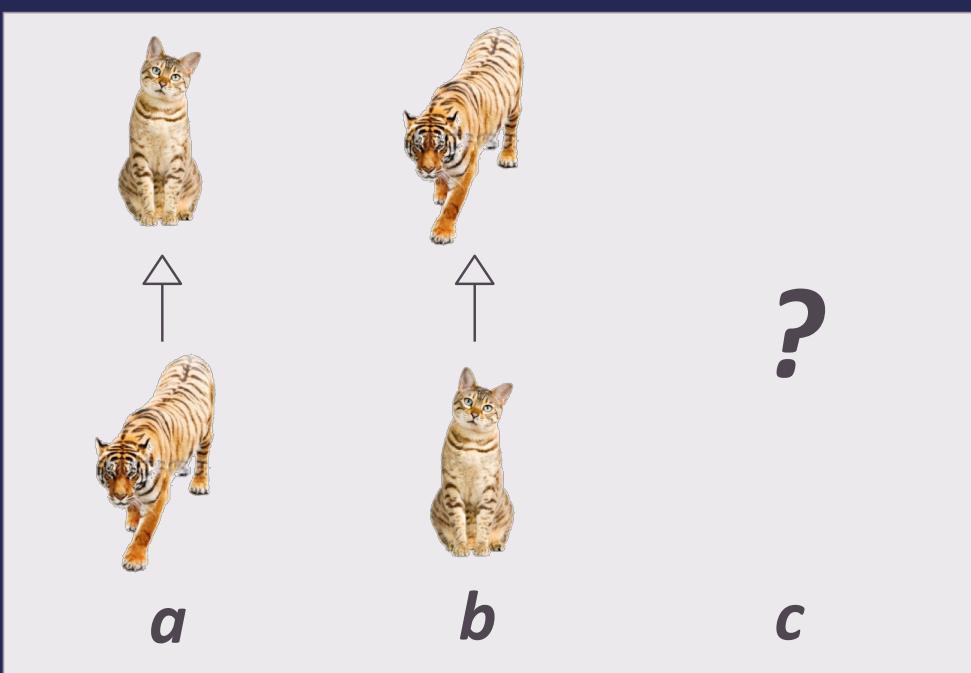
38

فایده‌های وراثت

- جلوگیری از نوشتن کد تکراری
- یک قرارداد مشترک برای گروهی از کلاس‌ها
(بعداً بیشتر خواهیم دید)

39

کدامیک درست است؟



40



inheritance hierarchy

سلسله مراتب وراثت

41

طراحی درخت وراثت برای برنامه شبیه‌سازی حیوانات

designing for inheritance

Let's design the inheritance tree for an Animal simulation program

Imagine you're asked to design a simulation program that lets the user throw a bunch of different animals into an environment to see what happens. We don't have to code the thing now, we're mostly interested in the design.

We've been given a list of *some* of the animals that will be in the program, but not all. We know that each animal will be represented by an object, and that the objects will move around in the environment, doing whatever it is that each particular type is programmed to do.

And we want other programmers to be able to add new kinds of animals to the program at any time.

First we have to figure out the common, abstract characteristics that all animals have, and build those characteristics into a class that all animal classes can extend.

1 Look for objects that have common attributes and behaviors.

What do these six types have in common? This helps you to abstract out behaviors. (Step 2)

How are these types related? This helps you to define the inheritance tree relationships (step 4-5)

170 chapter 7

Your Brain on Java—A Learner's Guide
2nd Edition - Covers Java 5.0
Head First Java
Kathy Sierra & Bert Bates

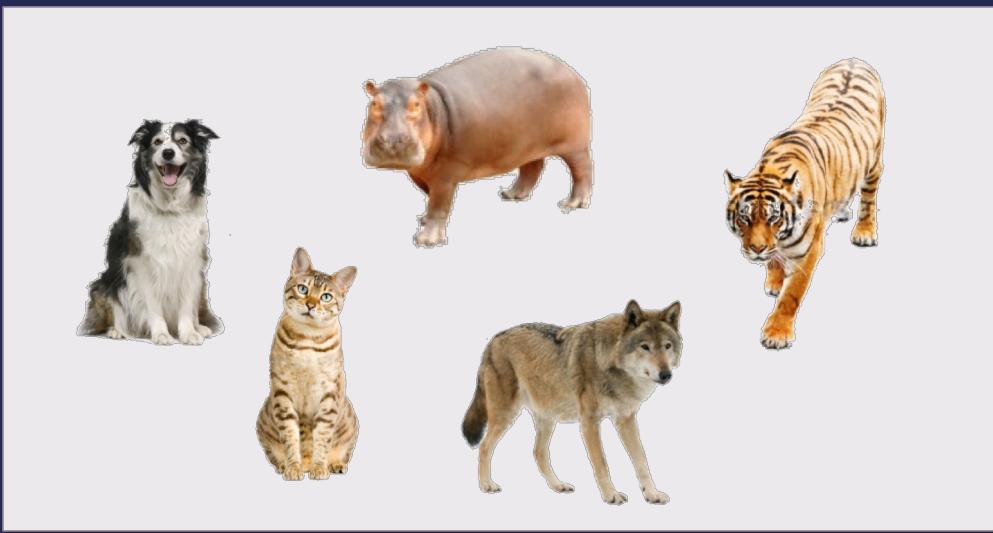
A book cover for "Your Brain on Java—A Learner's Guide" by Kathy Sierra and Bert Bates, 2nd Edition, covering Java 5.0. The cover features a woman with glasses pointing towards the viewer. The title "Head First Java" is prominently displayed in the center.

این مثال با تغییراتی برگرفته شده از



42

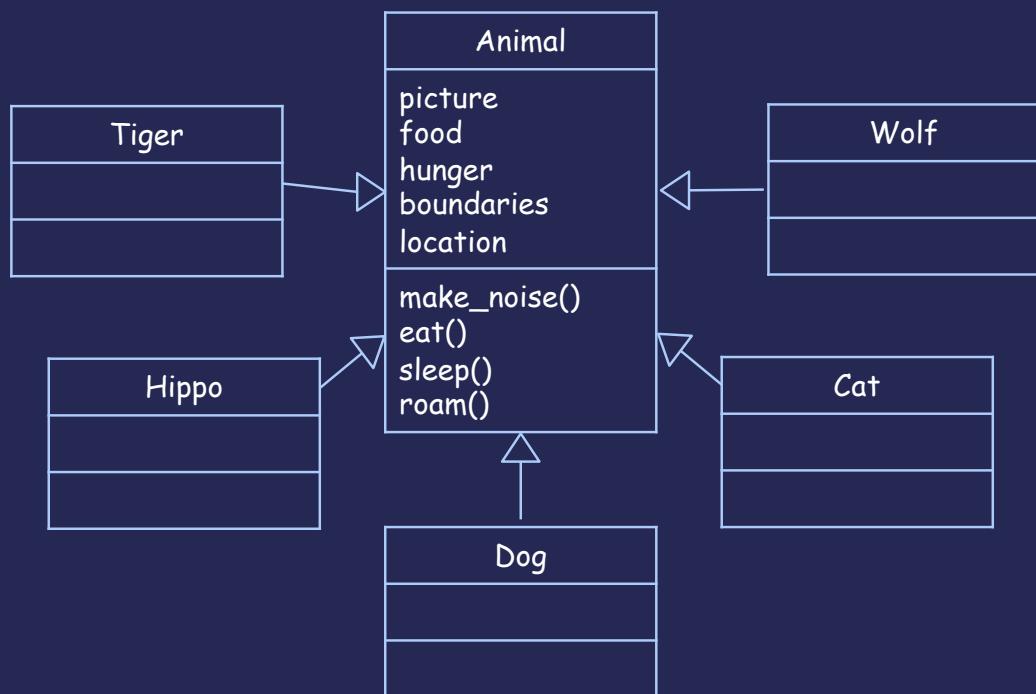
طراحی درخت وراثت برای برنامه شبیه‌سازی حیوانات



۱ به دنبال اشیائی می‌گردیم که صفات و رفتار مشابهی دارند

43

۲ کلاسی تعریف می‌کنیم که صفات و رفتار مشترک را مدل کند



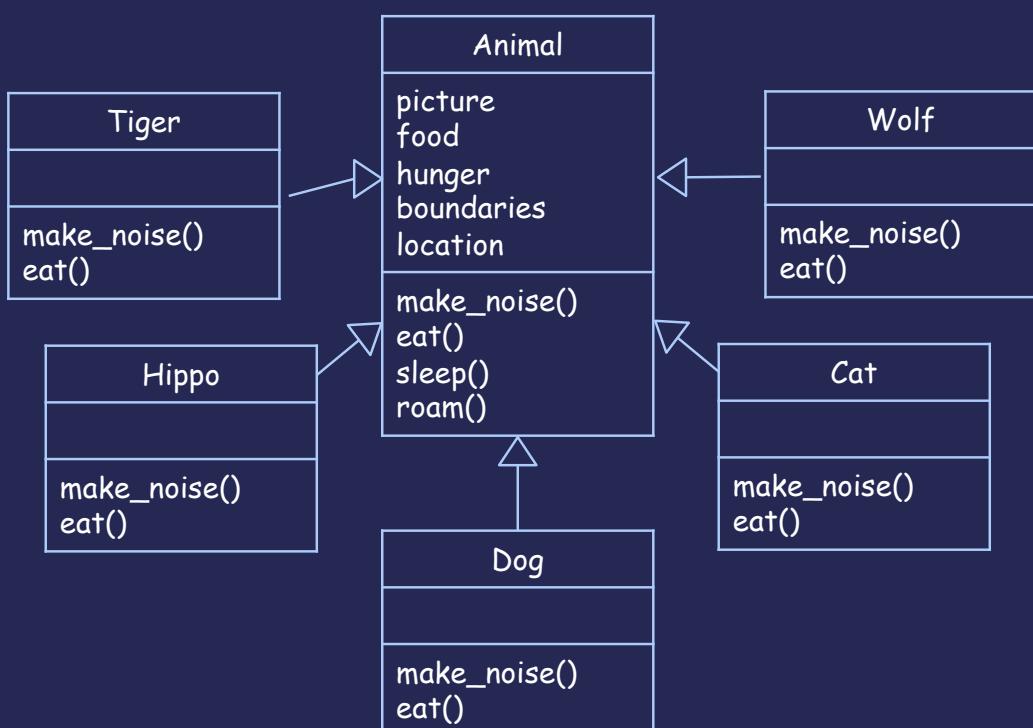
44

آیا هیچ یک از زیر کلاس ها رفتاری مختص خود دارد؟



45

بازنویسی متدهای `eat` و `make_noise` توسط زیر کلاس ها



46

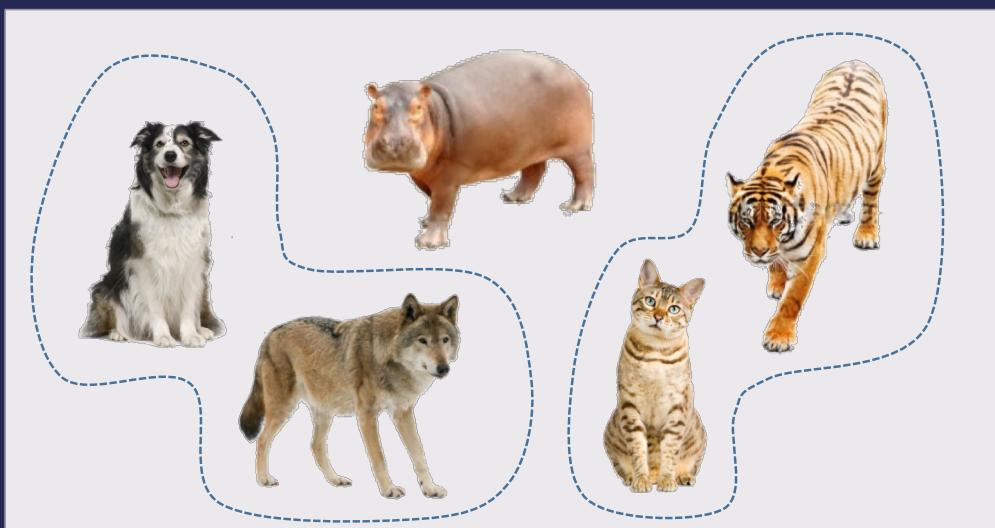
۴ در پی موقعیت‌هایی بیشتری برای تجرید هستیم. به دنبال دو یا چند زیرکلاس می‌گردیم که رفتار مشترک داشته باشند.



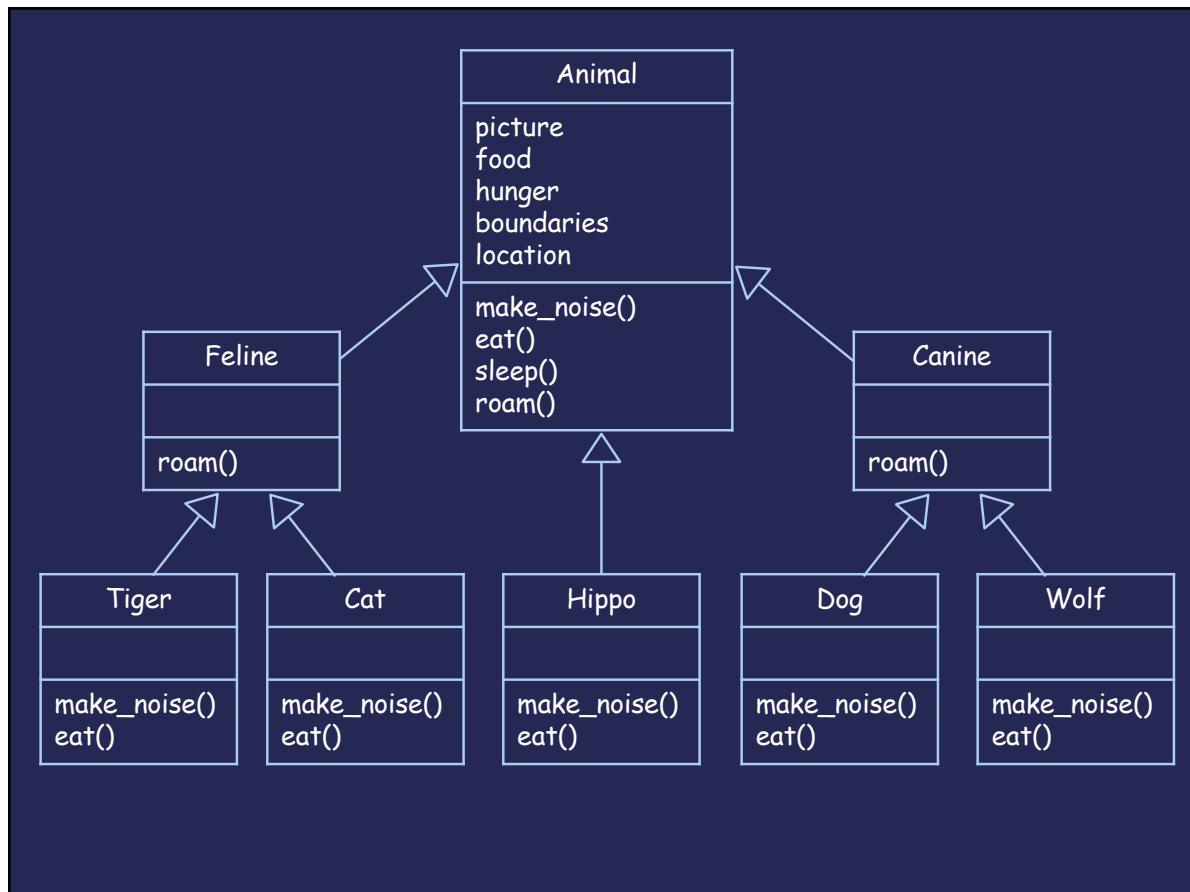
47

۴ در پی موقعیت‌هایی بیشتری برای تجرید هستیم. به دنبال دو یا چند زیرکلاس می‌گردیم که رفتار مشترک داشته باشند.

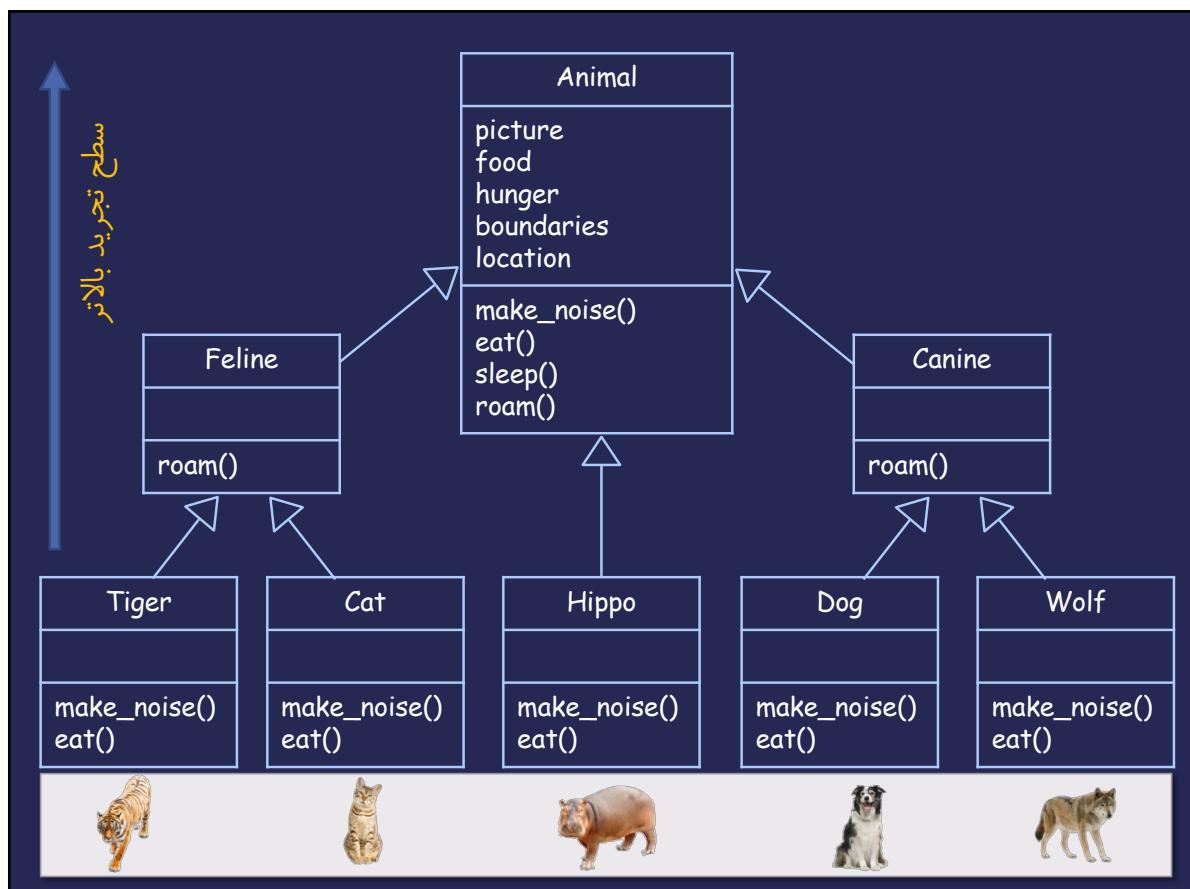
گربه و ببر سعی می‌کنند هنگام حرکت از همنوعان خود دوری کنند.
سگها و گرگها به صورت گلهای حرکت می‌کنند.



48



49

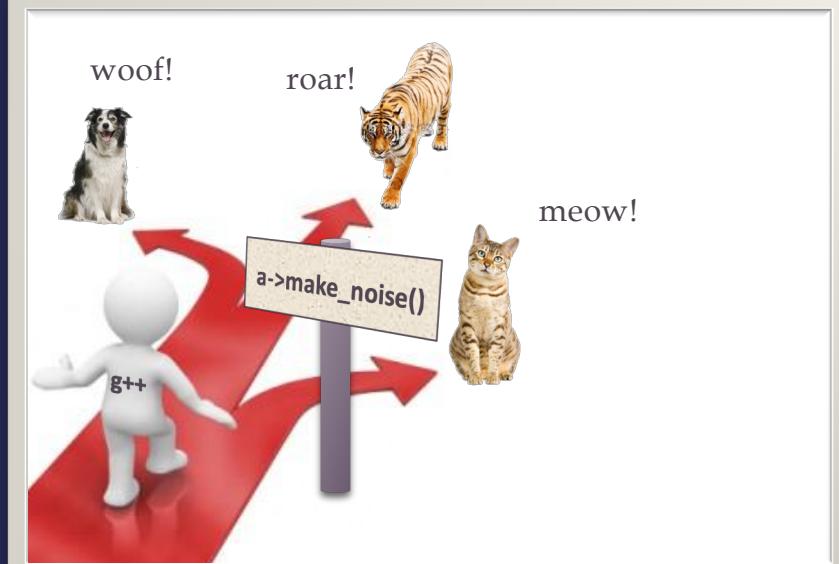


50

اگر شیرها هم در شبیه‌سازی ما باشند چه؟



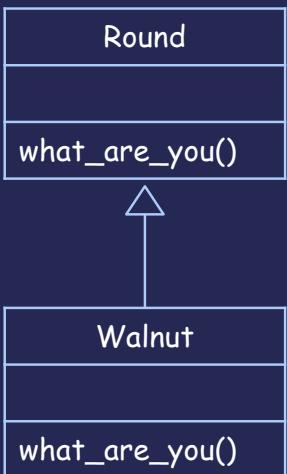
51



وابسته‌سازی پویا

52

گردو گرد است.



53

```
Round r;
Round* rp;
rp = &r;
cout << rp->what_are_you();
```



۱. آیا $rp = \&r$ مجاز است؟

۲. آیا $rp->what_are_you()$ مجاز است؟

۳. تشخیص کدی که در ازای $rp->what_are_you()$ باید صدا شود.

54

```

Round r;
Round* rp;
rp = &r;
cout << rp->what_are_you();

```



۱. آیا $rp = \&r$ مجاز است؟

با توجه به تایپ r و rp مجاز است (هر دو از نوع

۲. آیا $rp->what_are_you()$ مجاز است؟

۳. تشخیص کدی که در ازای $rp->what_are_you()$ باید صدا شود.

55

```

Round r;
Round* rp;
rp = &r;
cout << rp->what_are_you();

```



۱. آیا $rp = \&r$ مجاز است؟

با توجه به تایپ r و rp مجاز است (هر دو از نوع

۲. آیا $rp->what_are_you()$ مجاز است؟

با نگاه کردن به تایپ rp (آیا کلاس Round متدهی به نام what_are_you بدون پارامتر دارد؟)

۳. تشخیص کدی که در ازای $rp->what_are_you()$ باید صدا شود.

56

```

Round r;
Round* rp;
rp = &r;
cout << rp->what_are_you();

```



۱. آیا $rp = \&r$ مجاز است؟

با توجه به تایپ r و rp مجاز است (هر دو از نوع $(Round^*$

۲. آیا $rp->what_are_you()$ مجاز است؟

با نگاه کردن به تایپ rp (آیا کلاس $Round$ متدهای نام $what_are_you$ بدون پارامتر دارد؟)

۳. تشخیص کدی که در ازای $rp->what_are_you()$ باید صدا شود.

با نگاه کردن به تایپ rp (فراخوانی $what_are_you$ از کلاس $(Round$

57

```

Walnut w;
Round* rp;
rp = &w;
cout << rp->what_are_you();

```



۱. آیا $rp = \&w$ مجاز است؟

۲. آیا $rp->what_are_you()$ مجاز است؟

۳. تشخیص کدی که در ازای $rp->what_are_you()$ باید صدا شود.

58

```

Walnut w;
Round* rp;
rp = &w;
cout << rp->what_are_you();

```



۱. آیا $rp = \&w$ مجاز است؟

با توجه به تایپ w و rp مجاز است چون $Walnut$ زیرکلاس $Round$ است.

۲. آیا $rp->what_are_you()$ مجاز است؟

۳. تشخیص کدی که در ازای $rp->what_are_you()$ باید صدا شود.

59

```

Walnut w;
Round* rp;
rp = &w;
cout << rp->what_are_you();

```



۱. آیا $rp = \&w$ مجاز است؟

با توجه به تایپ w و rp مجاز است چون $Walnut$ زیرکلاس $Round$ است.

۲. آیا $(rp->what_are_you())$ مجاز است؟

با نگاه کردن به تایپ rp (آیا کلاس $Round$ متدهای نام $what_are_you$ بدون پارامتر دارد؟)

۳. تشخیص کدی که در ازای $rp->what_are_you()$ باید صدا شود.

60

```

Walnut w;
Round* rp;
rp = &w;
cout << rp->what_are_you();

```



۱. آیا $rp = \&w$ مجاز است؟

با توجه به تایپ w و rp مجاز است چون Walnut زیر کلاس Round است.

۲. آیا $rp->what_are_you()$ مجاز است؟

با نگاه کردن به تایپ rp (آیا کلاس Round متدهی به نام what_are_you بدون پارامتر دارد؟)

۳. تشخیص کدی که در ازای $rp->what_are_you()$ باید صدا شود.

با نگاه کردن به تایپ شیء اشاره شده توسط rp (فراخوانی (Walnut از کلاس what_are_you

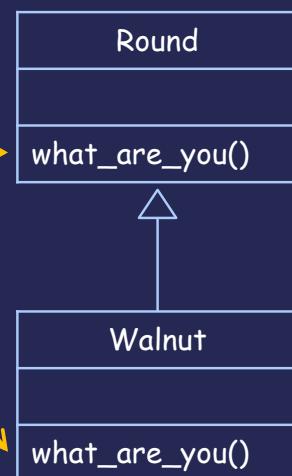
61

وابسته سازی – Binding

```

Round* rp;
...
rp->what_are_you();

```



این که فراخوانی مربوطه به کدامیک از متدها وابسته می‌شود بستگی به شیئی دارد که rp به آن اشاره می‌کند.

آیا می‌توان این را در زمان کامپایل تشخیص داد؟

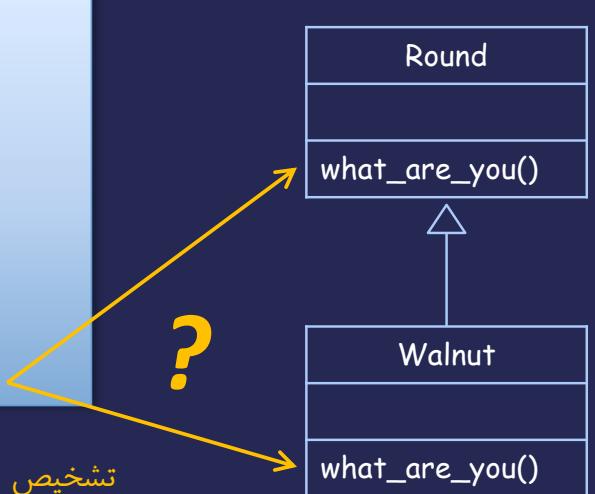
62

وابسته‌سازی پویا – Dynamic Binding

```
Round* rp;
Round r;
Walnut w;

if (rand()%2)
    rp = &r;
else
    rp = &w;

rp->what_are_you();
```



تشخیص تایپ شیئی که rp به آن اشاره می‌کند در زمان کامپایل ممکن نیست!

63

متدهای مجازی – Virtual Methods

وابسته‌سازی پویا فقط برای متدهای مجازی اتفاق می‌افتد.

```
class Round {
public:
    virtual string what_are_you() {
        return "gerd";
    }
};

class Walnut : public Round {
public:
    virtual string what_are_you() {
        return "gerdoo";
    }
};
```

64

متدهای مجازی – Virtual Methods

هنگام بازنویسی یک متدهای مجازی، لازم نیست کلمه‌ی `virtual` را مجدداً ذکر کنیم.

Once virtual,
always virtual!

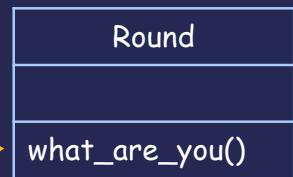
```
class Round {  
public:  
    virtual string what_are_you() {  
        return "gerd";  
    }  
};  
  
class Walnut : public Round {  
public:  
    string what_are_you() {  
        return "gerdoo";  
    }  
};
```

65

وابسته‌سازی ایستا – Static Binding

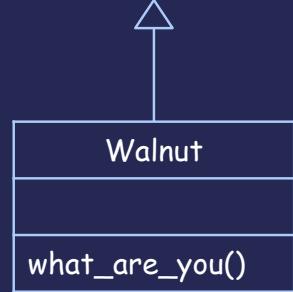
اگر متدهای `what_are_you` تعریف نشود:

```
Walnut w;  
Round* rp = &w;  
rp->what_are_you();
```



مستقل از شیئی که `rp` به آن اشاره می‌کند، متدهای فراخوانی توسط تایپ `rp` تعیین می‌شود.

قابل تشخیص در زمان کامپایل



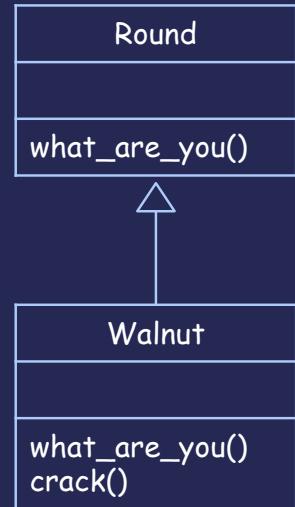
66

هر گردي گردو نیست!

```
Round r;  
Walnut* wp;  
wp = &r; ← compile error!
```

ممكن است با گردو کاري بکنيم
که با هر گردي نمی توانيم بکنيم.

```
wp->crack();
```



67

فایده‌ی وابسته‌سازی پویا چیست؟



68

با وابسته‌سازی پویا

```
class Vet {  
public:  
    void give_shot(Animal* a) {  
        // do horrible things to a!  
        a->make_noise();  
    }  
};  
  
int main() {  
    Dog snoopy;  
    Cat garfield;  
    Vet ghamar;  
  
    ghamar.give_shot(&snoopy);  
    ghamar.give_shot(&garfield);  
}
```

با فرض مجازی بودن make_noise کلاس Dog صدا می‌شود
کلاس Cat صدا می‌شود

69

بدون وابسته‌سازی پویا

```
class Vet {  
public:  
    void give_shot_to_dog(Dog* a) {  
        // do horrible things to a!  
        a->make_noise();  
    }  
    void give_shot_to_cat(Cat* a) {  
        // do horrible things to a!  
        a->make_noise();  
    }  
};  
int main() {  
    ...  
    ghamar.give_shot_to_dog(&snoopy);  
    ghamar.give_shot_to_cat(&garfield);  
}
```

70

بدون وابسته‌سازی پویا

```
class Vet {  
public:  
    void give_shot_to_dog(Dog* a) {  
        // do horrible things to a!  
        a->make_noise();  
    }  
    void give_shot_to_cat(Cat* a) {  
        // do horrible things to a!  
        a->make_noise();  
    }  
    // give_shot methods for other animals  
};
```

برای هر یک از انواع حیوانات یک متدهای جداگانه

بدنهای همه‌ی آنها مشابه است.

با اضافه شدن هر نوع حیوان باید متدهای مربوط به آن در کلاس Vet اضافه شود.

71

با وابسته‌سازی پویا ...

حتی برای انواع جدیدی از حیوانات
که بعداً تعریف خواهیم کرد!

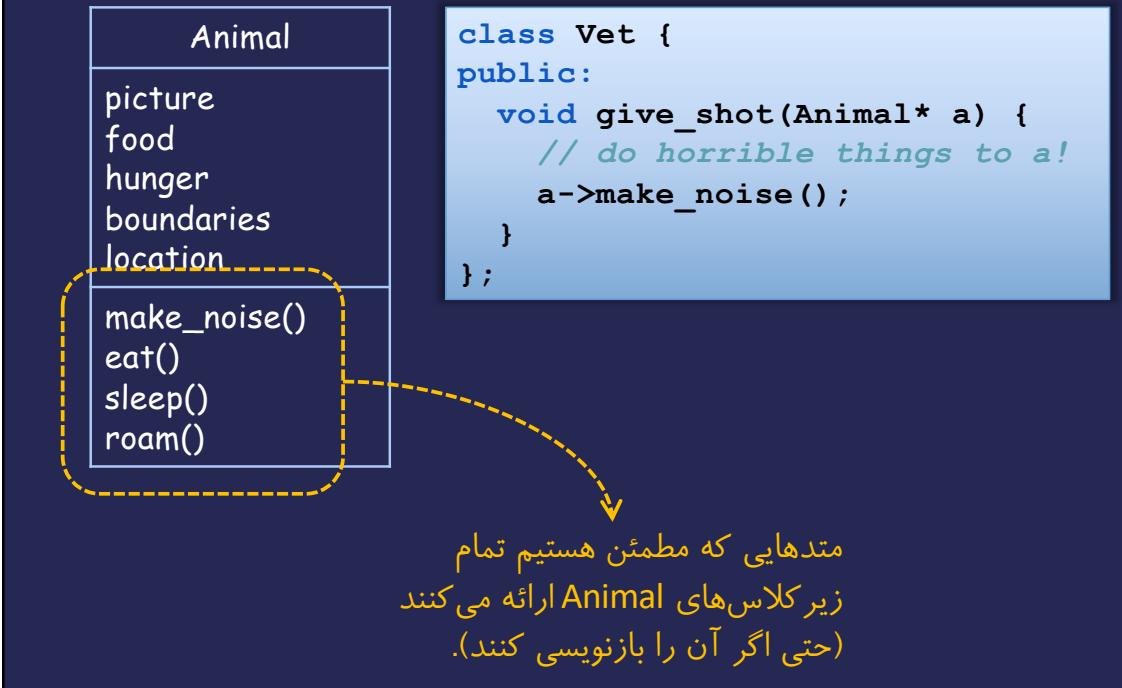
Guaranteed to work with
any type of animal!



مطمئن هستیم a->make_noise() کار می‌کند.
چون کلاس a یا make_noise را تعریف کرده یا از Animal به ارث برده.

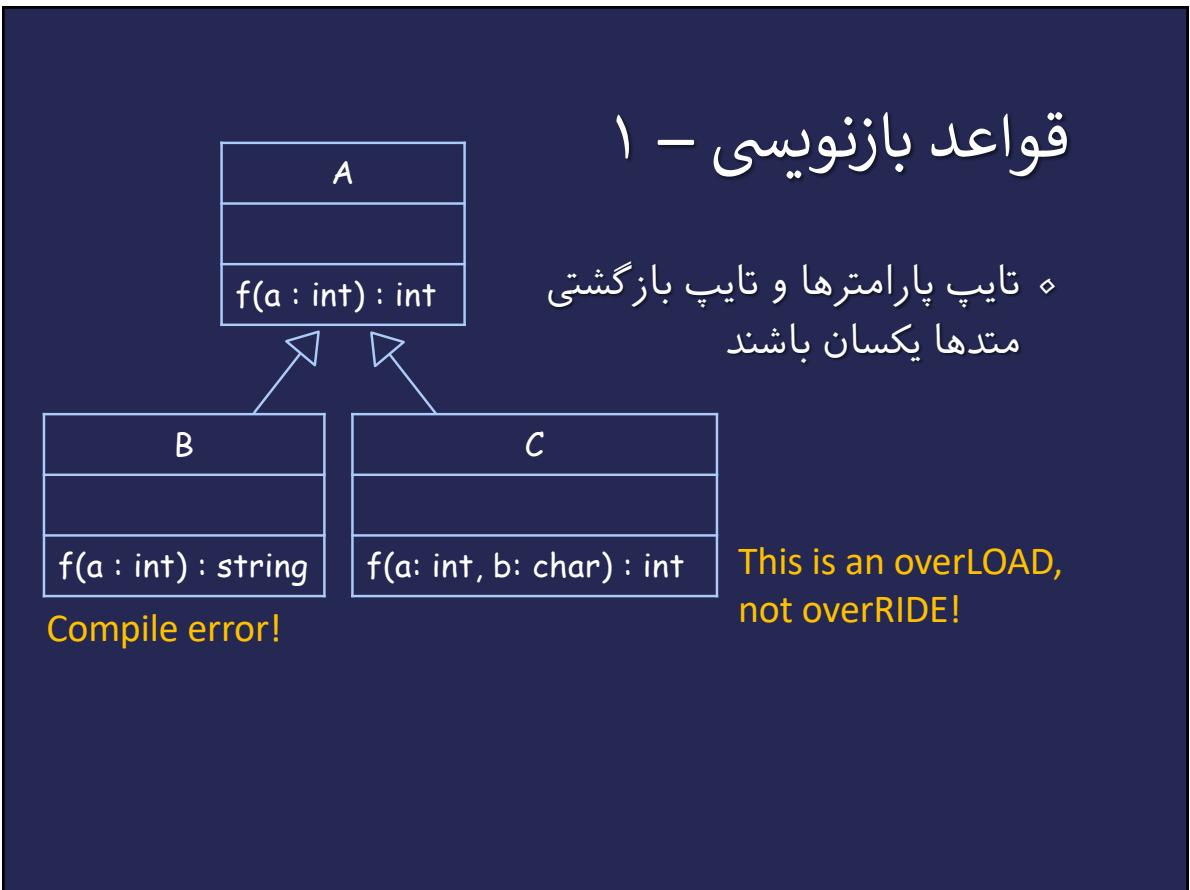
72

یک قرارداد مشترک برای گروهی از کلاس‌ها



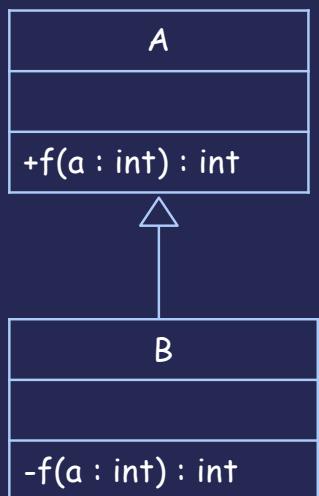
73

قواعد بازنویسی - ۱



74

قواعد بازنویسی - ۲



Compile error!

- ❖ سطح دسترسی متد زیرکلاس نباید ضعیفتر باشد.

```
B b;  
A* ap = &b;  
ap->f(2);
```

قرار بوده متد `B::f` را کسی از بیرون صدا نکند!

این اسلاید در C++ غلط و در جاوا درست است!

75

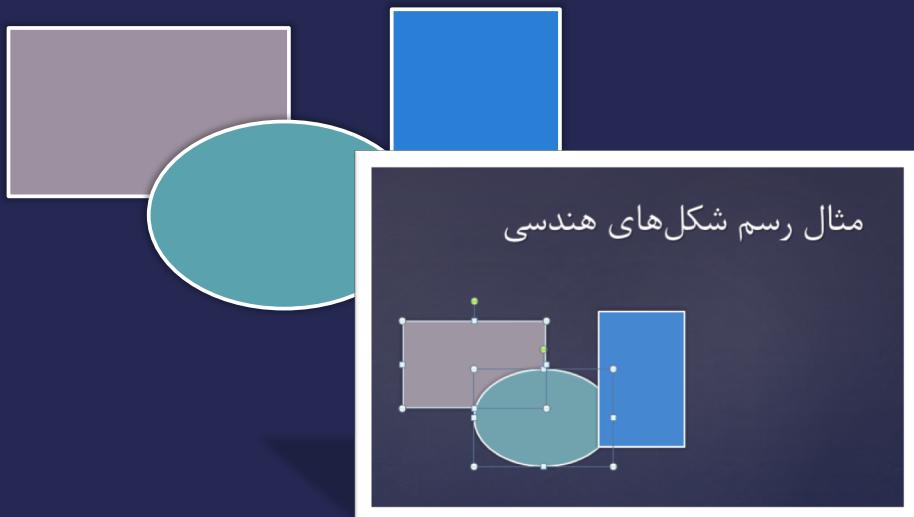


polymorphism

چندریختی

76

مثال رسم شکل‌های هندسی



77

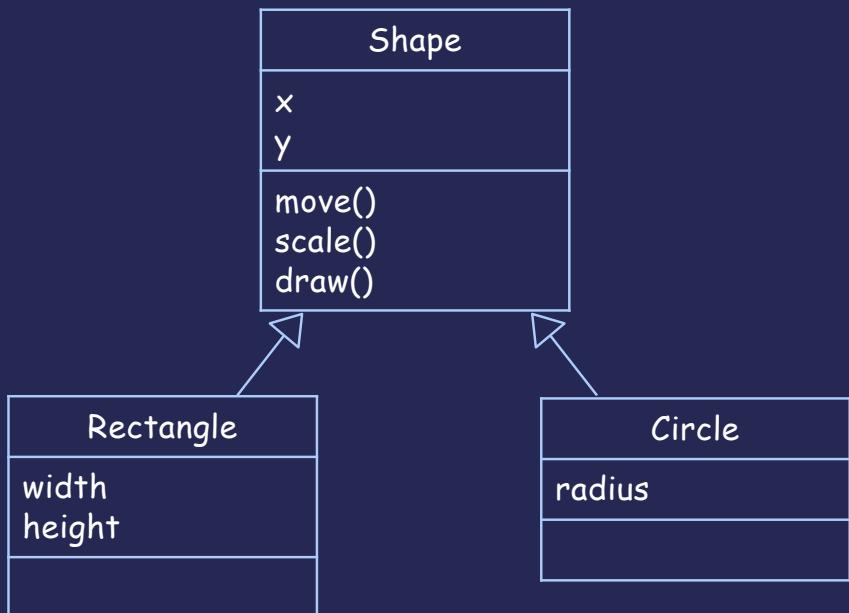
کلاس‌های متناظر با شکل‌ها

Rectangle
x
y
width
height
move()
scale()
draw()

Circle
x
y
radius
move()
scale()
draw()

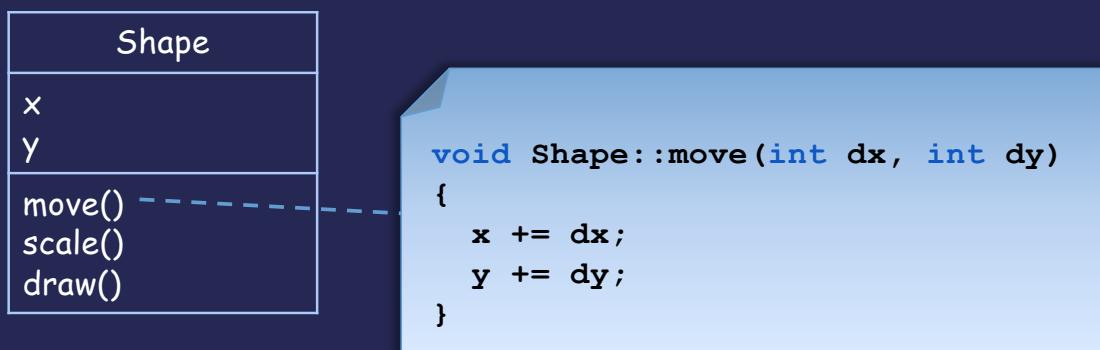
78

فاکتورگیری اعضای مشترک



79

پیادهسازی move



80

پیاده‌سازی scale

Shape
x
y
move() scale() - - - - - draw()



پیاده‌سازی این متد در کلاس Shape قابل انجام نیست و باید توسط زیرکلاس‌ها صورت‌پذیرد.

81

پیاده‌سازی scale

Shape
x
y
move() scale() - - - - - draw()

```
void Shape::scale(double factor)
{
    // do nothing
}
```

نظیر همین موضوع در مورد draw وجود دارد.

82

استفاده از Shape

```
Shape s(10, 20);
s.move(-1, 2);
s.scale(3);    // does nothing
```

اصلًاً یک شکل چه شکلی است؟!



83

کلاس‌های مجرد (abstract)

Shape
x
y
move()
scale()
draw()

```
class Shape {
public:
    Shape(int, int);
    void move(int dx, int dy);
    virtual void scale(double s) = 0;
    virtual void draw() = 0;
};
```

کلاس مجرد:
کلاسی که حداقل یک متد
مجازی خالص داشته باشد.

: (pure virtual)
متدهای خالص که بدن ندارد و پیاده‌سازی آن به
زیرکلاس‌ها واگذار می‌شود.

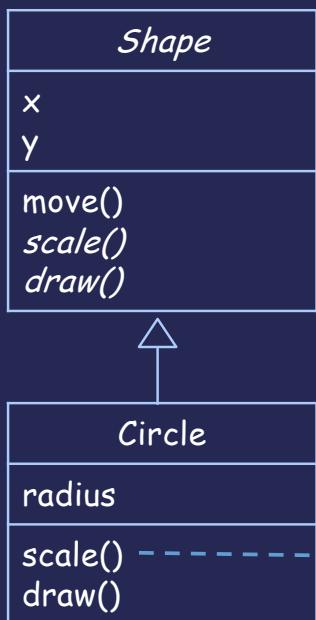
84

ساختن شیء از کلاس مجرد ممکن نیست!

```
Shape s(10, 20); ← compile error!
s.move(-1, 2);
s.scale(3);    // does nothing
```

85

مسئولیت زیرکلاس‌ها



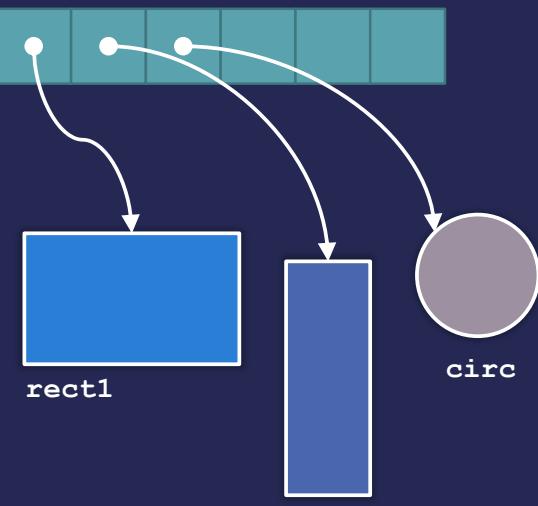
زیرکلاس‌ها باید متدهای مجرد را پیاده‌سازی کنند،
و گرنه خود نیز مجرد محسوب می‌شوند.

```
void Circle::scale(double factor)
{
    radius *= factor;
}
```

86

نگهداری شکل‌ها

shapes



```
vector<Shape*> shapes;
```

```
Rect rect1(10, 7, 3, 2);  
Rect rect2(3, 15, 1, 4);  
Circle circ(3, 1);  
  
shapes.push_back(&rect1);  
shapes.push_back(&rect2);  
shapes.push_back(&circ);
```

مشکل: لزوماً نمی‌دانیم کاربر برنامه چندتا مستطیل و چندتا دایره قرار است رسم کند.

87

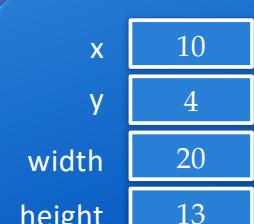
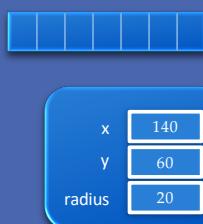
تخصیص حافظه‌ی پویا

```
Rect* rp = new Rect(...);  
...  
delete rp;
```

Stack

rp

Heap



88

آزاد کردن حافظه‌ی تخصیص‌یافته

- چرا باید حافظه‌ای را که به طور پویا می‌گیریم حتماً آزاد کنیم؟
- استفاده بهینه از حافظه در طول اجرا
- اتکا نکردن به سیستم عامل برای آزاد کردن حافظه پس از اتمام اجرا

89

انضباط در مدیریت حافظه

- حافظه‌ای را که به طور پویا گرفته‌اید را حتماً آزاد کنید.
- حافظه‌ای را که با `new` گرفته‌اید با `delete` و حافظه‌ای را که با `malloc` گرفته‌اید با `free` آزاد کنید
- حافظه‌ای را دوبار آزاد نکنید!

90