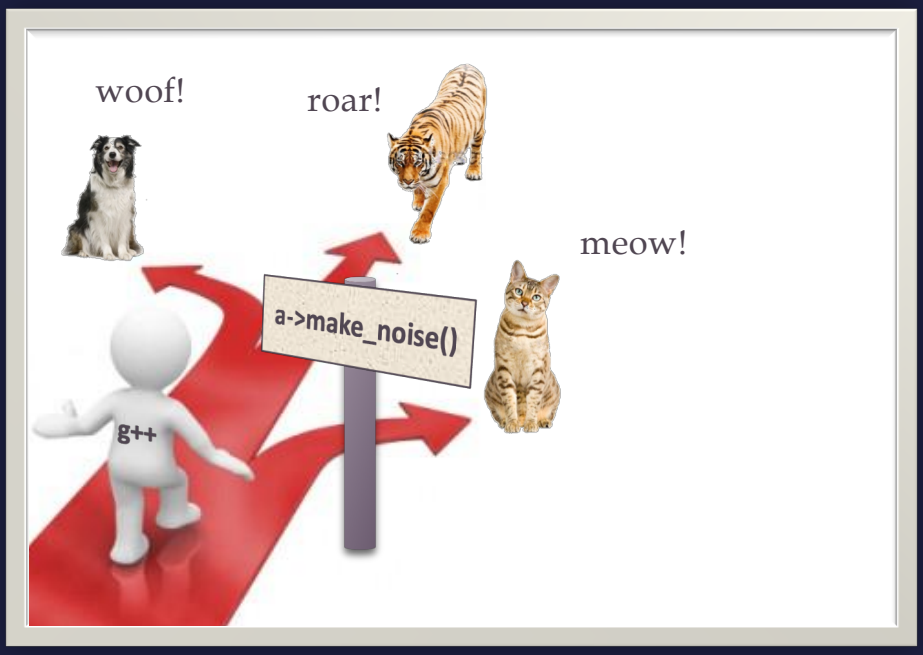
دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر

وراثت و چندریختی — ۵
مقیدسازی پویا

برنامه‌سازی پیشرفته — رامتین خسروی

بهار ۹۹

1

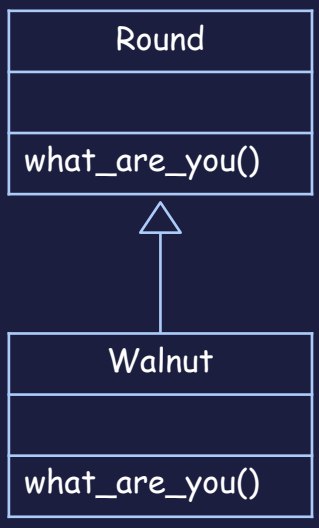


dynamic binding

مقیدسازی پویا

2

گردو گرد است.




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

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



```
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.

```
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؟

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


```
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 متدی به نام what_are_you بدون پارامتر دارد؟)

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

```
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؟

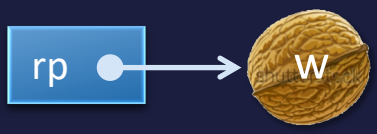
با نگاه کردن به تایپ rp (آیا کلاس Round متدی به نام what_are_you بدون پارامتر دارد؟)

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

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


```
Walnut w;

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



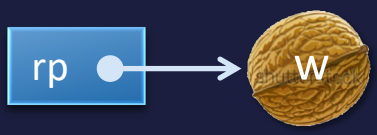
۱. آیا rp = &w مجاز است؟

۲. آیا rp->what_are_you() مجاز است؟

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

```
Walnut w;

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



۱. آیا rp = &w مجاز است؟

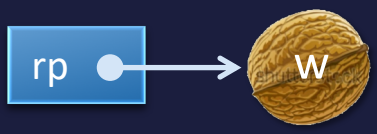
با توجه به تایپ w و rp مجاز است چون Walnut زیرکلاس Round است.

۲. آیا rp->what_are_you() مجاز است؟

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


```
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.

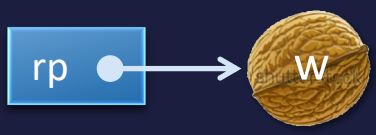
۲. آیا rp->what_are_you() مجاز است؟

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

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

```
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؟

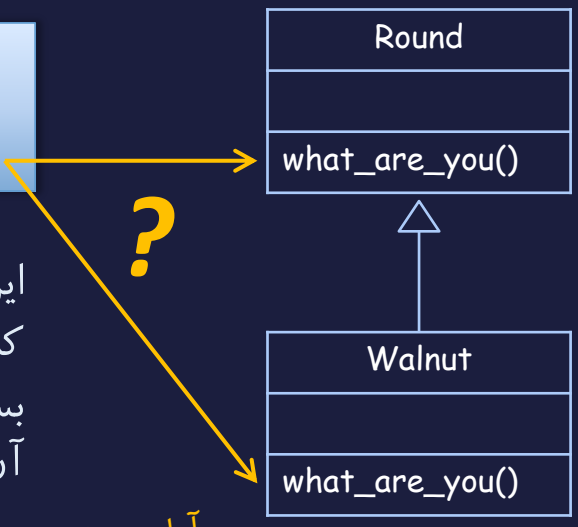
با نگاه کردن به تایپ rp (آیا کلاس Round متدی به نام what_are_you بدون پارامتر دارد؟)

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

با نگاه کردن به تایپ شیء اشاره‌شده توسط rp (فراخوانی what_are_you از کلاس Walnut)

مقیدسازی – Binding

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



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

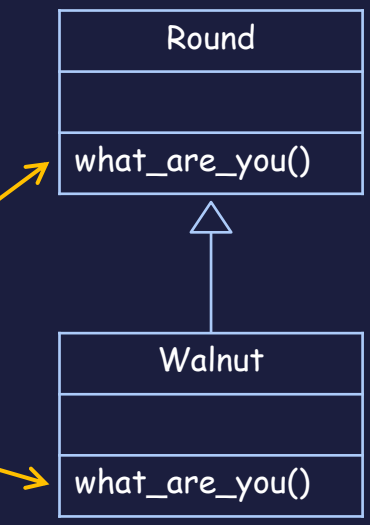
آیا می‌توان این را در زمان کامپایل تشخیص داد؟

Dynamic Binding – مقیدسازی پویا

```
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!

Virtual Methods – متدهای مجازی

مقیدسازی 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";
  }
};
```

Virtual Methods – متدهای مجازی

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

Once virtual, always virtual!

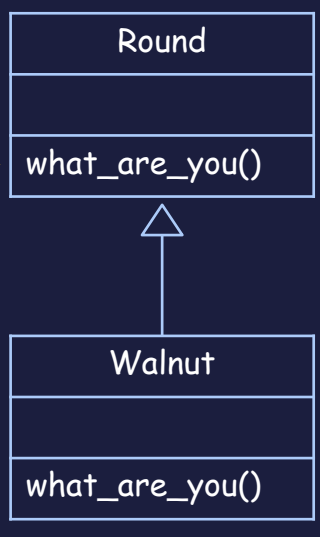
```
class Round {
public:
  virtual string what_are_you() {
    return "gerd";
  }
};

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


Static Binding – وابسته‌سازی ایستا

اگر متد what_are_you مجازی تعریف نشود:

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



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

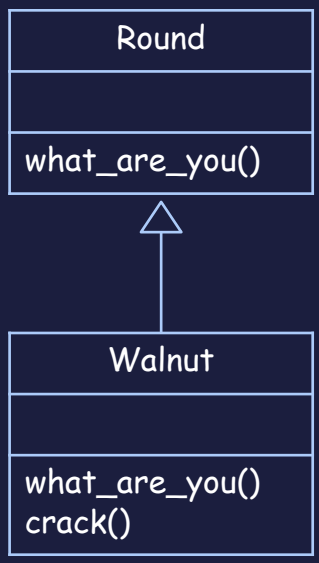
قابل تشخیص در زمان کامپایل

هر گردی گردو نیست!

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

ممکن است با گردو کاری بکنیم
که با هر گردی نمی‌توانیم بکنیم.

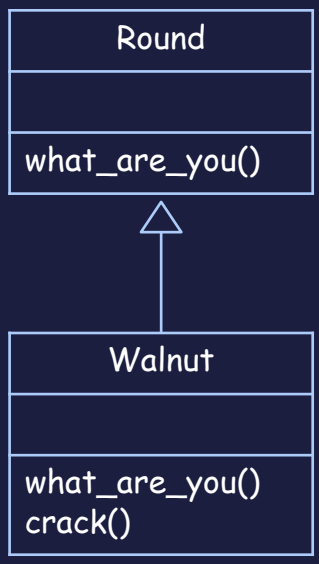
```
wp->crack();
```



تبدیل تایپ‌ها

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

rp = &w;        // up-casting
```

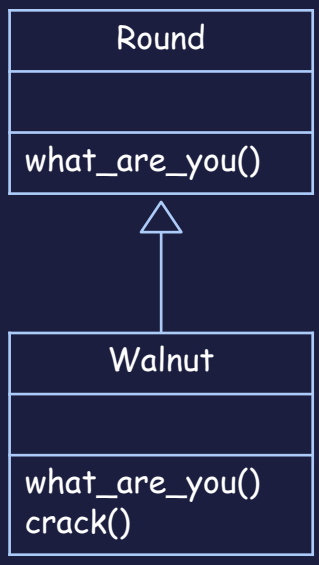


تبدیل تایپ‌ها

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

rp = &w;       // up-casting

// unsafe down-casting:
wp = (Walnut*)rp;
```

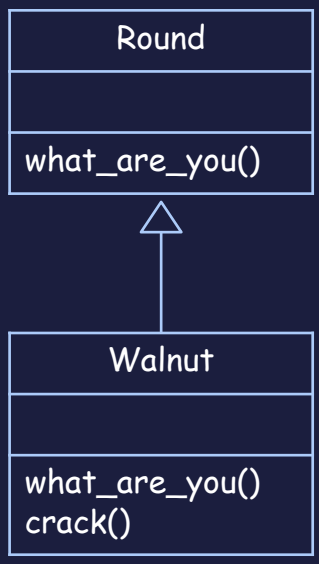


تبدیل تایپ‌ها

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

rp = &w;       // up-casting

// safe down-casting:
wp = dynamic_cast<Walnut*>(rp);
if (wp == NULL)
    cerr << "Sorry, not a Gerdoo!";
else
    wp->crack(); // Safe to crack!
```



برش‌زدن اشیاء

```
class Base {
    int a;
};

class Derived : public Base {
    int b;
};

…
Base b;
Derived d;

b = d;
…
```


تمرین

خروجی این برنامه را بدون اجرا کردن آن مشخص کنید.

```
class superbase {
public:
    void f() { cout << 1; }
    virtual void g() { cout << 2; }
    void h() { g(); }
};

class base : public superbase {
public:
    virtual void g() { cout << 3; }
};

class derived : public base {
public:
    void f() { cout << 4; }
};
```

```
int main() {
    superbase s;
    base b;
    derived d;
    superbase* p;

    p = &s;
    p->f();
    p->g();
    p->h();

    p = &b;
    p->f();
    p->g();
    p->h();

    p = &d;
    p->f();
    p->g();
    p->h();
}
```