Polymorphism
چندریختگی

معرفی

- چندریختگی جنبه دیگری از جداسازی رابط ها از پیاده سازی را جهت تفاوت قائل شدن بین دو مفهوم چه(what) و چگونه(how) فراهم می کند.
- از طریق چندریختگی کدهای بهبود یافته سازمندهی شده و خواناتر می شوند و همچنین اجازه ایجاد برنامه های توسعه پذیر جدید را نیز می دهد که این برنامه ها می توانند چه در طول ایجاد پروژه اصلی و چه در هنگامی که ویژگی جدیدی مدنظرمان هست توسعه یابند.

Upcasting

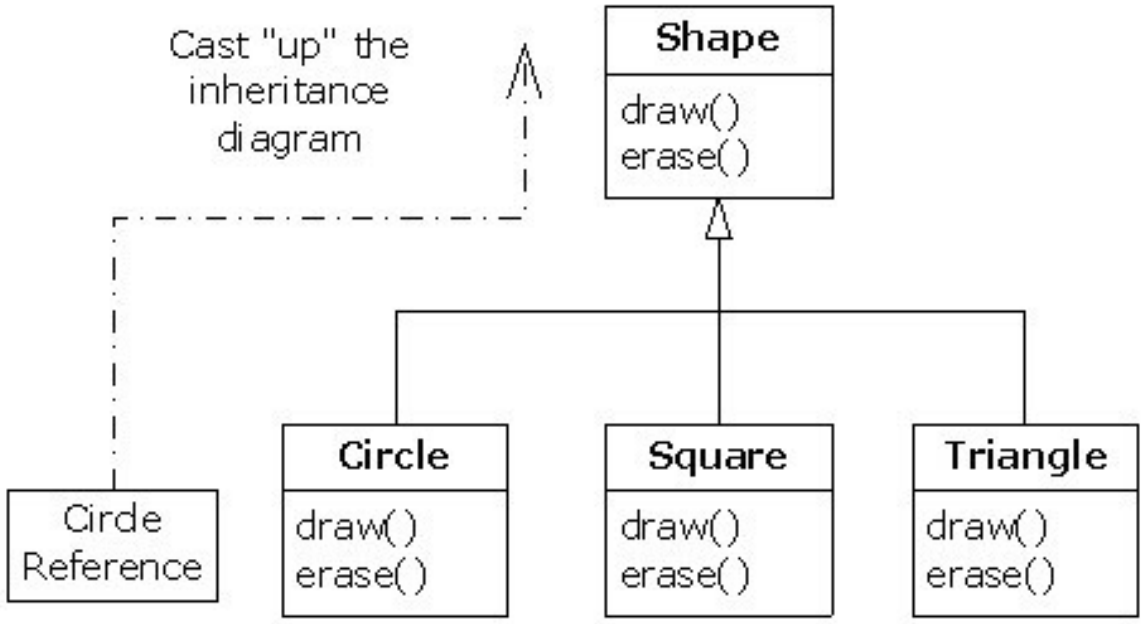
- عملیاتی که در آن referenceای از یک objectرا گرفته و با آن مانند referenceای به پایه رفتار می شود را upcasting می گویند.چرا که شیوه ای که درخت ارث بری نمایش داده می شود به این صورت است که کلاس پایه در بالای درخت قرار دارد.
- از طریق upcasting یک شی از جنس کلاس مشتق شده به کلاس اصلی خود تبدیل می شود.

Method-call binding

- اتصال فراخوانی یک متد به بدنه آن متد،انقیاد (binding) نامیده می شود.
- هنگامی که عملیات binding قبل از اجرای برنامه صورت گیرد(توسط کامپایلر و linker،در صورت وجود) به آن Early binding می گویند.
- **Late binding به این معناست که binding بسته به نوع object در زمان اجرا صورت گیرد و به آن dynamic binding و runtime binding.نیز می گویند**
- جاوا از مکانیزم Late Binding استفاده می کند و اتصال تمام متدها ، چه static باشند و چه از نوع final ، در زمان اجرا صورت می گیرد.(متدهای private به صورت ضمنی از نوع final هستند.)
  - ✓ این به این معناست که معمولا احتیاجی به تصمیم گیری درباره چگونگی و تعیین زمان late binding ندارید چرا که عملیات مربوط به آن به صورت خودکار انجام می شود.

Method-call binding

- با در نظرگرفتن این که تمام متدهای جاوا به صورت چندریختی از طریق late bindig اتفاق می افتند،در واقع پیغامی را به یک object ارسال می کنیم و به object اجازه محاسبات لازم جهت انجام کارهایش به درستی را می دهیم.(مثال:کلاس کارخانه)



خطر: Override کردن متدهای private

```
public class PrivateOverride {
   private void f() { print("private f()"); }
   public static void main(String[] args) {
      PrivateOverride po = new Derived();
      po.f();
  }
}
class Derived extends PrivateOverride {
   public void f(){
    print("public f()");
 }
}
```

- نتیجه این عملیات این است که تنها متدهای override، non-private می شوند.البته باید در رابطه با شیوه انجام override متدهای private باید احتیاط کرد چرا که ممکن است عملیات مد نظرمان را انجام ندهد و کامپایلر هم هیچ warning ای را ندهد.

Constructorها و چندریختی

✓شیوه فراخوانی های constructor

1. Constructor کلاس پایه فراخوانی می شود.این مرحله به صورت بازگشتی تکرار می شود.به شیوه ای که ابتدا ریشه سلسله مراتب ساخته می شود و سپس اولین کلاس مشتق شده و تا زمانی که به بیشترین کلاس های مشتق شده دست یابیم،ادامه می یابد .
2. مقداردهنده های اعضا(Member initializerها) به ترتیب اعلان،فراخوانی می شوند.
3. بدنه constructor کلاس مشتق فراخوانی می گردد.

رفتار متدهای چندریختی در داخل constructor ها

- در صورتی که در داخل constructorای باشید و یک متد از object ساخته شده را فراخوانی کنید،چه اتفاقی می افتد؟
- در صورتی که در داخل constructor یک متد را فراخوانی کنید،تعریف override شده ای از آن متد را استفاده کرده اید.

رفتار متدهای چندریختی در داخل constructor ها

```
class Glyph {
  void draw() { print("Glyph.draw()"); }
   Glyph() {
         print("Glyph() before draw()");
         draw();
         print("Glyph() after draw()");
  }
 }

 class RoundGlyph extends Glyph {
    private int radius = 1;
   RoundGlyph(int r) {
     radius = r;
    print("RoundGlyph.RoundGlyph(), radius = " + radius);
  }
void draw() {
    print("RoundGlyph.draw(), radius = " + radius);
  }
 }
public class PolyConstructors {
   public static void main(String[] args) {
     new RoundGlyph(5);
  }
}
```

رفتار متدهای چندریختی در داخل constructor ها

- فرآیند واقعی مقداردهی اولیه به صورت زیر است:

1. حافظه تخصیص یافته به object، قبل از هر چیزی با مقدار صفر باینری مقداردهی اولیه می شود.
2. Constructor کلاس پایه به شیوه ای که در اسلایدهای قبل توضیح داده شد،فراخوانی می شود. در این نقطه متدoverride شدهdraw ()فراخوانی می‌شود
3. (قبل از آنکه costructor کلاس RoundGlyph فراخوانی شود.) و با مقدار صفر (با توجه به مرحله اول) مواجه می شود.
4. مقداردهنده های اعضا(Member initializerها) به ترتیب اعلان،فراخوانی می شوند.
5. بدنه constructor کلاس مشتق فراخوانی می گردد.

نوع برگشتی Covariant

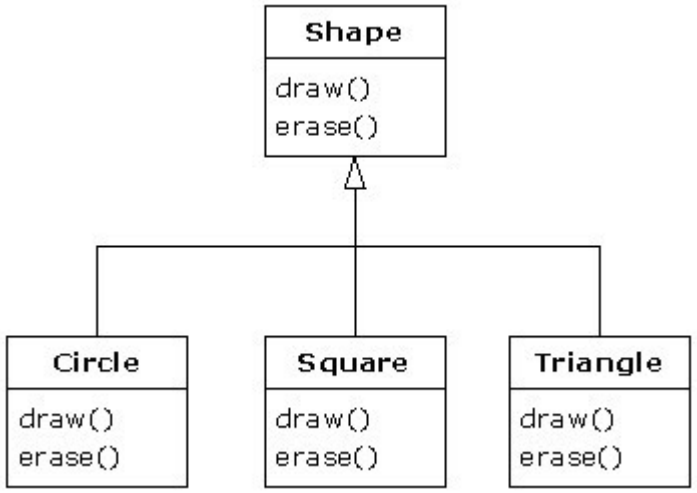
Covariant :تبدیل نوع های داده از یک حالت وسیع تر به حالتی جزئی تر در موقعیت های خاص

- در جاوا SE5 نوع برگشتی covariant اضافه شده است و به این معناست که متد override شده در کلاس مشتق می تواند نوعی را مشتق شده از نوع برگشتی متدهای کلاس های پایه،برگرداند.

```
class Grain {
  public String toString() { return "Grain"; }
}
class Wheat extends Grain {
  public String toString() { return "Wheat"; }
}
class Mill {
  Grain process() { return new Grain(); }
}
class WheatMill extends Mill {
  Wheat process() { return new Wheat(); }
}
```

جانشین سازی در مقایسه با گسترش

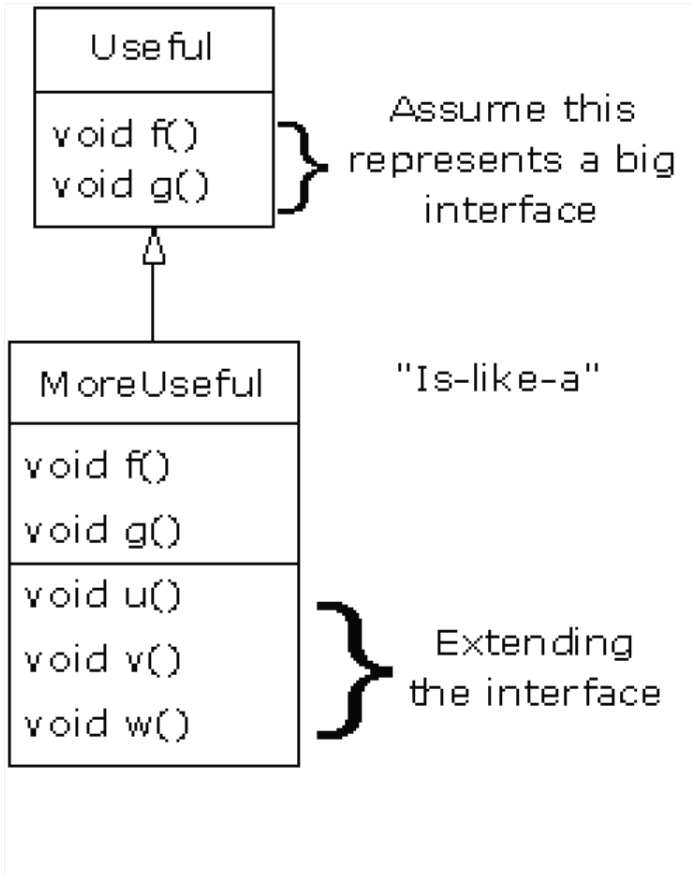
- بهترین و ایمن ترین راه برای ایجاده سلسه مراتب ارث بری بکار گیری یک شیوه جامع است.
- این شیوه را می توان رابطه جامع "(is-a)" نامید چرا که interface کلاس است که تعیین می کند چه چیزی هست.
- کلاس های مشتق هم بیشتر از interface کلاس پایه نخواه



جانشین سازی در مقایسه با گسترش

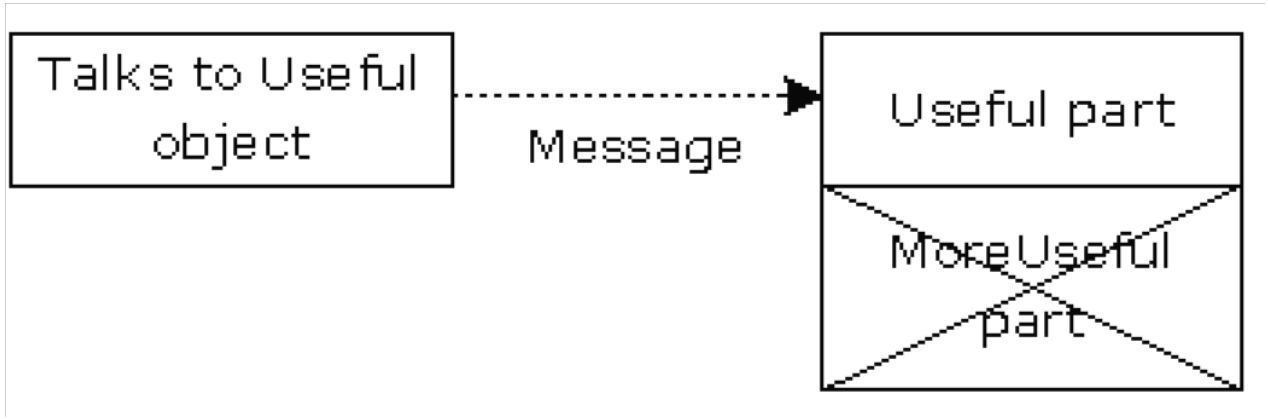
- به نظر می رسد رابطه جامع "is-a" تنها راه معقولانه برای انجام کارهاست و سایر شیوه های طراحی دارای تفکری مبهم و تعاریف ناقص هستند.همچنین این شیوه نوعی ترفند است.بااین طرز فکر دیدگاه شما نیز تغییر خواهد کرد و می توان رابط را گسترش داد.
- می توان این شیوه را رابطه " is-like-a" نامید چرا که کلاس مشتق مشابه(like) کلاس پایه است.

جانشین سازی در مقایسه با گسترش



جانشین سازی در مقایسه با گسترش

- اغلب با حالتی روبرو می شوید که در آن احتیاج به مشخص کردن دوباره نوع واقعی object برای دسترسی به متد های مشتق شده با آن نوع دارید.در شکل زیر چگونگی انجام این کار نشان داده شده است:



Downcasting و runtime type information (RTTI)

- برای حرکت به پایین سلسله مراتب ارث بری از downcast استفاده می شود.
- یک upcast همیشه امن است چراکه کلاس پایه نمی تواند interface ای بزرگتر از کلاس مشتق داشته باشد.اما downcast امن نیست.

```
/*Syntax : */ ((MoreUseful)x).u();
```

- در کلاس جاوا هر cast چک می شود و در صورتی که صحیح نبود آن exceptionای از ClassCastException دریافت خواهید کرد.

runtime type information (RTTI) و Downcasting

```
 class Useful {
   public void f() {}
   public void g() {}
}
class MoreUseful extends Useful     {
   public void f() {}
   public void g() {}
   public void u() {}
   public void v() {}
   public void w() {}
}
public class RTTI {
public static void main(String[] args) {
Useful[] x ={new Useful(), new  MoreUseful() };
    x[0].f();
    x[1].g();
((MoreUseful)x[1]).u();
// Downcast/RTTI
((MoreUseful)x[0]).u();
// Exception
 }
}
```