

# Reusing Classes

استفاده دوباره از کلاس ها



# معرفی

- می توان از طریق استفاده ی مجدد از کلاس ها به جای ایجاد آن ها از ابتدا، از کلاس های موجود که توسط افرادی build و debug شده، استفاده نمود.
- ترفند مدنظر این است که از کلاس های موجود بدون آن که کد آن ها را دستکاری کنیم استفاده نماییم.
- دو راه برای انجام این کار وجود دارد:
- ایجاد objectهایی از کلاس های موجود در داخل کلاس جدید، که به آن ترکیب (composition) می گویند.
- ایجاد یک کلاس جدید از نوع کلاس موجود، که به آن وراثت (inheritance) می گویند.

# قواعد ترکیب

□ شما به سادگی می توانید reference های object ها را در داخل کلاس های جدید قرار دهید

```
class WaterSource {  
    private String s;  
}  
public class SprinklerSystem {  
    private String valve1, valve2, valve3, valve4;  
    private WaterSource source = new WaterSource();  
}
```

□ اگر Reference های object ها با null مقداردهی اولیه شده اند و در صورتی که سعی کنید متدی را برای هر کدام از آن ها فراخوانی کنید، با خطای زمان اجرا روبرو خواهید شد.

# قواعد ترکیب

□ در صورتی که می خواهید Reference ها مقداردهی اولیه شوند، به صورت زیر عمل کنید:

✓ در نقطه ای که object ها تعریف شده اند

✓ در constructor آن کلاس

✓ درست قبل از آنکه احتیاج به استفاده از object باشد. به این تکنیک معمولا مقداردهی کند (lazy initialization) می گویند.

✓ از تکنیک مقداردهی اولیه نمونه (instance initialization) استفاده می کند.

# قواعد وراثت

□ قواعد ترکیب واضح و روشن است. اما برای انجام عملیات ارث بری فرمی کاملاً متفاوت لازم است. هنگامی که از مفهوم وراثت استفاده می کنیم، می گوییم "کلاس جدید مشابه کلاس پایه است" و این مفهوم را در کد، قبل از باز شدن براکت های بدنه کلاس، با استفاده از کلیدواژه extends و سپس نام کلاس پایه، قید می کنیم.

```
class Cleanser {  
    public void scrub() { append(" scrub()"); }  
}  
public class Detergent extends Cleanser {  
    public void scrub() {  
        append(" Detergent.scrub()");  
        super.scrub();  
    }  
}
```

# مقداردهی اولیه کلاس پایه

❑ از بیرون به نظر می رسد که کلاس جدید همان interface کلاس پایه را دارد و شاید تنها چند متد و فیلد به آن اضافه شده است. اما مفهوم وراثت تنها کپی کردن interface کلاس پایه نیست.

❑ وقتی object را از کلاس مشتق شده ایجاد می کنیم، در واقع درون آن حاوی یک subobject از کلاس پایه است.

❑ با صدا زدن constructor کلاس مربوطه constructor کلاس پدر که همه دانش و امتیازات لازم برای مقداردهی اولیه کلاس خود را دارد، نیز انجام می گیرد.

# Delegation

□ سومین رابطه که به طور مستقیم در جاوا پشتیبانی نمی شود  
Delegation نام دارد.

□ Delegation چیزی بین وراثت و ترکیب است. چرا که شما یک object  
عضو را در کلاسی که در حال ساخت آن هستید قرار می دهید (مثل  
عملیات ترکیب) و در عین حال در همان زمان تمام متدهای object  
عضو را در اختیار کلاس جدید قرار می دهید (همانند عملیات وراثت)  
□ به عنوان مثال یک فضاپیما نیاز به ماژول کنترلی دارد اما نمی  
توانیم بگوییم که فضاپیما یک ماژول کنترلی است.

# Delegation

```
public class SpaceShipControls {  
    void up(int velocity) {}  
    void down(int velocity) {}  
}  
public class SpaceShipDelegation {  
    private SpaceShipControls controls = new SpaceShipControls();  
    public void up(int velocity) {  
        controls.up(velocity); //Delegation  
    }  
    public void down(int velocity) {  
        controls.down(velocity); //Delegation  
    }  
}
```



# ترکیب در مقایسه با وراثت

□ ترکیب عموماً زمانی استفاده می شود که می خواهید از ویژگی های کلاس موجود در داخل کلاس جدید استفاده کنید اما رابط آن را نمی خواهید. این به این معنی است که object را قرار می دهید و از طریق استفاده از آن می توانید ویژگی های کلاس جدید را پیاده سازی کنید.

□ هنگامی که از وراثت استفاده می کنید، از کلاس موجود استفاده می کنید و ورژن جدیدی از آن را ایجاد می کنید. با اندکی فکر در رابطه با این موضوع در می یابید که این که یک «اتومبیل» را با «وسیله نقلیه» ترکیب کنید، معقول نیست. یک «اتومبیل» شامل «وسیله نقلیه» نیست بلکه یک خود یک «وسیله نقلیه» است.

□ Is-A رابطه ای است که مفهوم وراثت را بیان می کند

□ Has-A. رابطه ای است که مفهوم ترکیب را بیان می کند

# protected


این به این معنی است که private است تا آن جا که به کاربر کلاس مربوط می شود. اما در دسترس برای هر کلاسی که از این کلاس ارث می برد و هر کس دیگر در همان package می باشد. (در جاوا مفهوم protected دسترسی package را نیز فراهم می کند).

# Upcasting


مهم ترین جنبه ارث بری این نیست که متدهایی را برای کلاس جدید ایجاد کنیم. "کلاس جدید خود نوعی از کلاس موجود است."

```
class Instrument {  
    public void play() {}  
    static void tune(Instrument i) {  
        i.play();  
    }  
}  
  
public class Wind extends Instrument {  
    public static void main(String[] args) {  
        Wind flute = new Wind();  
        Instrument.tune(flute); // Upcasting  
    }  
}
```

# چرا از upcasting استفاده کنیم؟

Upcasting همیشه امن است چرا که از یک نوع خاص تر به یک نوع 

عمومی تر می‌رسیم

ممکن است کلاس مشتق شده متدهای بیشتری را داشته باشد اما باید 

حداقل، متدهای موجود در کلاس پایه را داشته باشد.

# کلیدواژه final

❑ بسیاری از زبان های برنامه نویسی روشی دارند که از طریق آن به کامپایلر می گویند داده ای ثابت (constant) است.

❑ استفاده از constant به دو دلیل مفید است:

- می تواند ثابتی در زمان کامپایل باشد که هیچگاه تغییر نمی کند.
- می تواند مقداری باشد که در زمان جهت مقداردهی اولیه استفاده می شود و هیچ گاه تغییر نمی کند.

❑ فیلدی که هم static و هم final باشد تنها یک نقطه از حافظه را دارد که هیچ گاه نمی تواند تغییر کند.

# کلیدواژه final

□ استفاده از final مقداری را constant می کند.

□ در رابطه با reference، استفاده از final، آن را constant می کند.

□ Blank Final فیلدهایی هستند که به عنوان، final تعریف شده اند . اما مقدار اولیه ای نگرفته اند

✓ در این موارد شما مجبور به تخصیص final ها با دستوری در نقطه تعریف و یا در هر constructor می باشید.

# کلیدواژه final

```
class Value {  
    int i; // Package access  
    final int o ; //Blank Final  
    public Value(int i) { this.i = i;  o = i * 2; }  
}  
public class FinalData {  
    private static Random rand = new Random(47);  
    private final int valueOne = 9;  
    private static final int VALUE_TWO = 99;  
    public static final int VALUE_THREE = 39;  
    private final int i4 = rand.nextInt(20);  
    static final int INT_5 = rand.nextInt(20);  
    private final Value v1 = new Value(11);  
    private final int[] a = { 1, 2, 3, 4, 5, 6 } ;  
}
```

# کلیدواژه final

□ آرگومان های final:

این به این معنی است که در داخل متد نمی توانید تغییری در reference آرگومان به چه چیزی اشاره می کند، اعمال کنید.

□ متدهای final:

بدین معنی است که در متد قفلی را قرار دهید تا از تغییر آن توسط کلاس های فرزند جلوگیری نمایید.



# کلیدواژه final

## کلاس های final:

✓ در مواردی شما نمی خواهید که کلاسی از کلاس مدنظرتان ارث ببرد و یا نمی خواهید کلاس مورد نظر تغییر کند.

✓ به بیانی دیگر، به دلایلی شیوه طراحی کلاستان به صورتی است که احتیاج به هیچ تغییری در آن نیست و یا به دلایل امنیتی نمی خواهید هیچ زیر کلاسی داشته باشید.

# مقداردهی اولیه و بارگذاری کلاس

□ به این نکته توجه داشته باشید که کد کامپایل شده برای هر کلاس در فایل جدای مربوط به آن وجود دارد و این فایل تا زمانی که به کد مورد نظر نیازی نباشد، بارگذاری نمی شود.

□ به طور کلی می توان گفت: “کد کلاس در اولین استفاده از آن اجرا می شود.” و این در واقع زمانی است که اولین object کلاس ساخته می شود.

□ البته هنگامی که یک فیلد static یا متد static در دسترس قرار می گیرد نیز، بارگذاری رخ می دهد.