

بخش ۱۰: Upcasting و Downcasting – هنر تغییر دید

تهیه شده توسط: سید سجاد پیراهاش

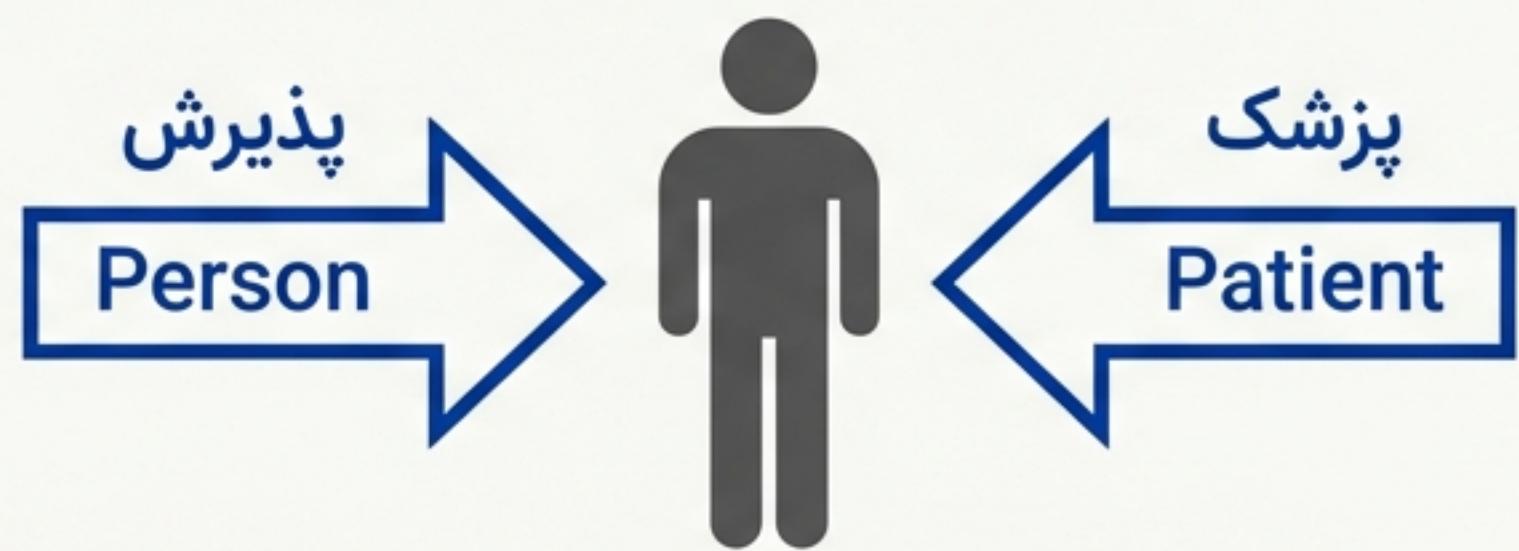


شما یک شیء هستید، اما با دیدگاه‌های متفاوت

تصور کنید به یک بیمارستان می‌روید. برای پذیرش، شما یک Person (شخص) هستید. اما برای پزشک، شما یک Patient (بیمار) هستید. هویت اصلی شما تغییر نمی‌کند، اما زمینه و تعاملات تغییر می‌کنند.

پزشک دیدگاه خود را از «شخص» به «بیمار» تغییر می‌دهد تا بتواند کارهای تخصصی (مانند تجویز تخصصی (مانند تجویز دارو) را انجام دهد.

این جوهره Casting در برنامه‌نویسی شیء‌گرا است.





نگاه از دور با تلسکوپ: Upcasting

یعنی گرفتن یک ارجاع از نوع فرزند و قرار دادن آن در یک متغیر از نوع والد. این عمل همیشه امن و ضمنی (Implicit) است، چون هر فرزند «هست یک» والد. جاوا این کار را به صورت خودکار انجام می‌دهد.

Examples:

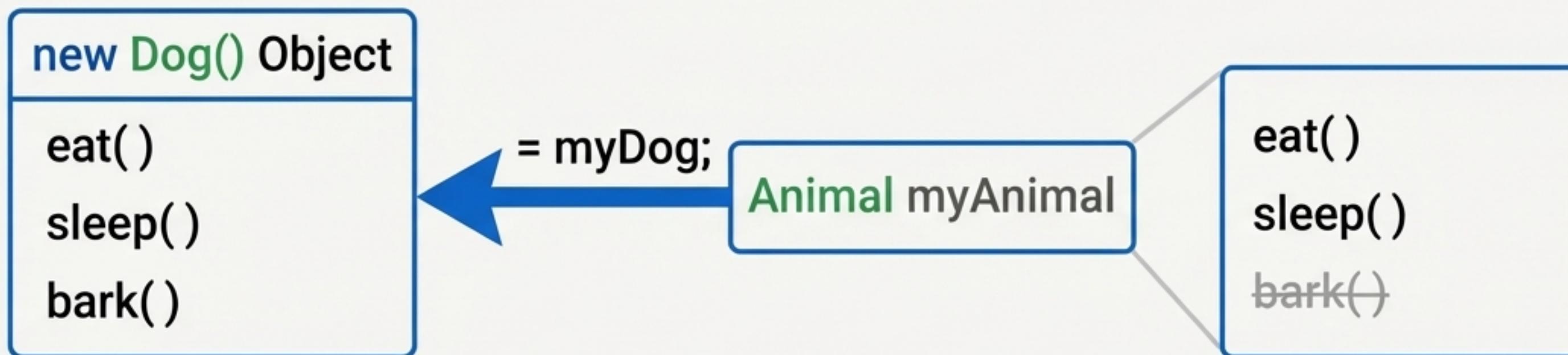
- `Dog` -> `Animal`
- `Manager` -> `Employee`

```
// Upcasting به صورت خودکار و ضمنی رخ می‌دهد
Dog myDog = new Dog();
Animal myAnimal = myDog; // ! امن!
```

چه چیزی به دست می‌دهیم؟ چه چیزی از دست می‌دهیم؟

وقتی یک شیء Dog را از طریق یک رفرنس Animal نگاه می‌کنید، به طور موقت توانایی دیدن متدهای اختصاصی bark() (مانند Dog) را از دست می‌دهید. شما فقط به متدهایی دسترسی دارید که در کلاس Animal تعریف شده‌اند.

Losing Details



چرا این محدودیت را می‌پذیریم؟ قدرت چندریختی! (Polymorphism)

به ما اجازه می‌دهد تا کدی بنویسیم که با انواع مختلفی از اشیاء به صورت یکپارچه کار کند. می‌توانیم مجموعه‌ای از Animal‌ها داشته باشیم که در آن هم Dog و هم Cat وجود دارد و با همه آنها به یک شکل رفتار کنیم.

```
Animal[] animals = new Animal[3];
animals[0] = new Dog(); // Upcasting
animals[1] = new Cat(); // Upcasting
animals[2] = new Dog(); // Upcasting
```

```
// یک حلقه برای همه حیوانات، بدون توجه به نوع واقعی آنها!
for (Animal a : animals) {
    a.eat(); // این قدرت چندریختی
}
```



نگاه از نزدیک با میکروسکوپ: Downcasting

عمل بر عکس است: گرفتن یک ارجاع از نوع والد و تلاش برای تبدیل آن به نوع فرزند. این عمل **بالقوه خطرناک** است و باید همیشه به صورت صریح (Explicit) انجام شود. شما باید به کامپایلر بگویید که از این تبدیل مطمئن هستید.

Examples:

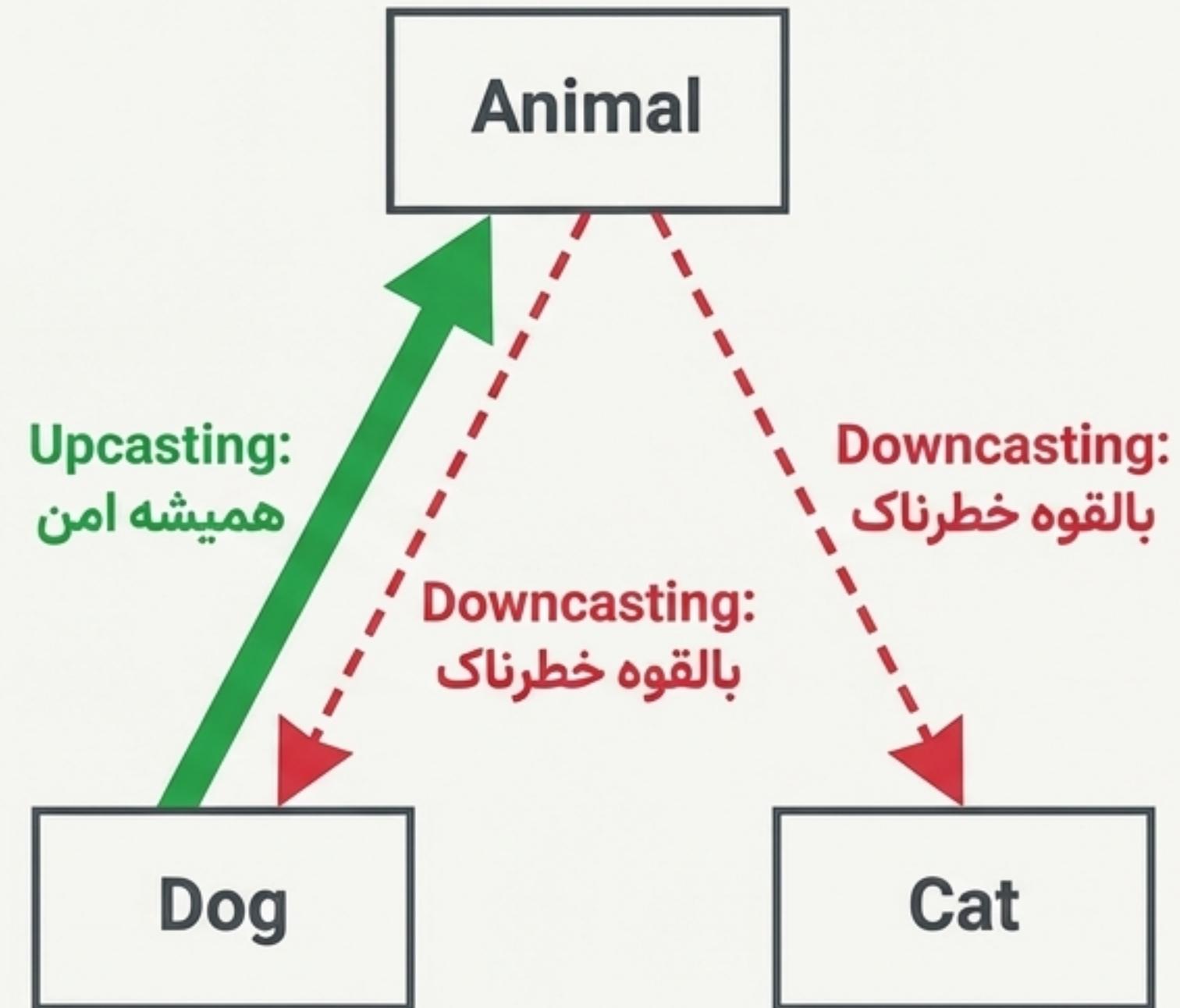
- `Animal` -> `Dog`
- `Employee` -> `Manager`

```
Animal myAnimal = new Dog(); // Upcast که Dog شده شروع با یک شیء
// برای دسترسی مجدد به bark()، باید `Dog` کنیم
Dog myDog = (Dog) myAnimal; // (Explicit Cast) نیاز به تبدیل صریح
myDog.bark();
```

⚠ خطر! اگر نمونه زیر میکروسکوپ اشتباه باشد...

چرا Downcasting خطرناک است؟
چون یک `Animal` لزوماً چون یک `Animal` لزوماً یک `Dog` نیست؛ ممکن است یک `Cat` باشد!
اگر به کامپایلر دروغ بگویید و سعی کنید یک `Cat` را به `Dog` تبدیل کنید، برنامه شما در زمان اجرا با خطای متوقف خواهد شد.

```
Animal myAnimal = new Cat();  
  
Dog myDog = (Dog) myAnimal;  
// BOOM! ClassCastException
```





تور نجات: قبل از زوم کردن، برچسب را چک کنید!

برای جلوگیری از `ClassCastException`، جاوا اپراتور `instanceof` را فراهم کرده است. این اپراتور قبل از انجام `Downcast`، چک می‌کند که آیا شیء مورد نظر واقعاً از آن نوع هست یا نه. این مانند یک «تست DNA» برای اشیاء است.

```
if (reference instanceof Type) {  
    // اینجا هستید، امن است  
}
```

همیشه، همیشه، همیشه `instanceof` از `Downcast` استفاده کنید.

کدنویسی امن: الگوی صحیح Downcasting

فرض کنید متدهای دارید که یک `Employee` دریافت می‌کند. می‌خواهیم اگر آن کارمند یک `Manager` بود، متدهای خاص (approveBonus()) را صدا بزنیم.

کد خطرناک

```
این کد ممکن است ClassCastException بیفت! //  
void performAction(Employee e) {  
    Manager m = (Manager) e;  
    m.approveBonus();  
}
```

کد امن و حرفه‌ای

```
الگوی صحیح با استفاده از instanceof  
void performAction(Employee e) {  
    if (e instanceof Manager) {  
        Manager m = (Manager) e; // این cast امن است  
        m.approveBonus();  
    }  
}
```

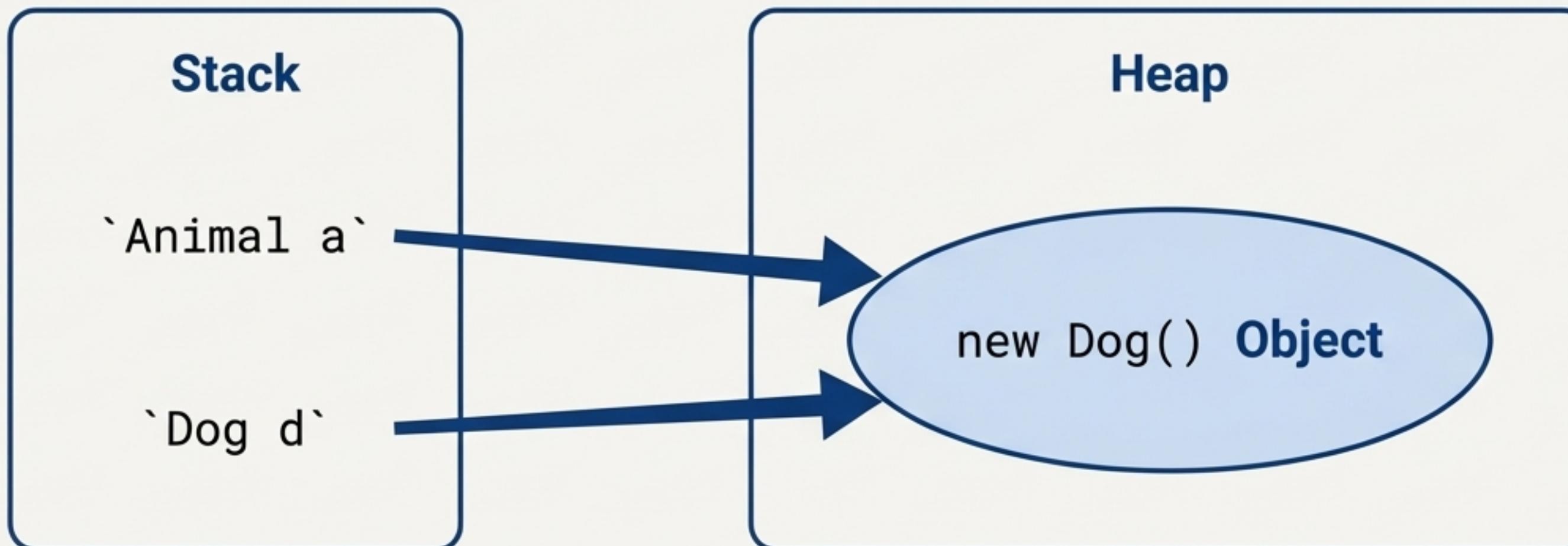
مقایسه رو در رو: Upcasting در مقابل Downcasting

معیار	Upcasting	Downcasting
تعريف	تبديل نوع از یک والد به یک فرزند.	تبديل نوع از یک فرزند به یک والد.
جهت	به سمت بالا در سلسله مراتب (خاص به عام).	به سمت پایین در سلسله مراتب (عام به خاص).
امنیت	همیشه امن (Safe)	بالقوه نامن (Unsafe)
نحوه اجرا	ضمنی (Implicit) Animal a = new Dog();	صريح (Explicit) Dog d = (Dog) a;
زمان بررسی	زمان کامپایل (Compile-time)	زمان اجرا (Runtime)
نتیجه خطا	خطای کامپایل	ClassCastException
دسترسی	محدود می شود: دسترسی به متدهای فرزند ممکن می شود.	گسترش می یابد: دسترسی به متدهای فرزند از بین می رود.
کاربرد اصلی	(Polymorphism)	دسترسی به رفتارهای خاص یک زیرکلاس.

دید کامپایلر در مقابل واقعیت در زمان اجرا

یک نکته حیاتی: Casting هرگز خود شیء را در حافظه Heap تغییر نمی‌دهد. شیء همان چیزی است که با `new` ساخته شده. فقط نوع «برچسب» یا «دید» متغیر ارجاعی (Reference) را در حافظه Stack تغییر می‌دهد.

- کامپایلر (Compile-time): فقط نوع رفرنس را می‌بیند. او به شما اجازه می‌دهد فقط متدهای تعریف شده در آن نوع رفرنس را صدا بزنید.
- Downcast (inRuntime): نوع واقعی شیء در Heap را می‌شناسد. او مسئول بررسی درستی ها و مسئول بروزرسانی درستی در JVM است. پرتاب `ClassCastException` است.



تله‌های رایج: چه زمانی استفاده پیش از حد از `instanceof` بُد است؟

اگر متدى دارید که پر از زنجیره‌های `if-else-if` با `instanceof` است، این معمولاً نشانه‌ای از طراحی ضعیف است. این یعنی شما به جای استفاده از قدرت چندریختی، در حال نوشتن کد رویه‌ای (Procedural) در یک محیط شی‌ءگرا هستید.

مثال از یک  Code Smell

```
void feed(Animal a) {  
    if (a instanceof Dog) {  
        // feed dog food  
    } else if (a instanceof Cat) {  
        // feed cat food  
    } else if (a instanceof Lion) {  
        // feed meat  
    }  
}
```

راه بهتر چیست؟ بازنویسی متدها (Method Overriding). این موضوع را در بخش بعدی بررسی خواهیم کرد!

چالش پیش‌بینی: این کد موفق می‌شود یا شکست می‌خورد؟

برای هر قطعه کد، پیش‌بینی کنید: ۱) خطای کامپایل ۲) خطای زمان اجرا (``Exception`)

```
Animal a = new Dog();  
Cat c = (Cat) a;
```

خطای زمان اجرا

```
Dog d = new Dog();  
Animal a = d;
```

اجرا موفق

```
Object o = "Hello";  
Integer i = (Integer) o;
```

خطای زمان اجرا

```
Animal a = new Animal();  
Dog d = (Dog) a;
```

خطای زمان اجرا

تمرین عملی: جزئیات خاص را به روشنی امن استخراج کنید

با استفاده از سلسله مراتب کلاس های `Publication` (والد)، `Book` (فرزند با متدهای `getAuthor()` و `getIssueNumber()`)، متدهای زیر را کامل کنید.

```
public void getSpecificDetails(Publication p) {  
    System.out.println("Checking publication...");  
  
    // کد شما اینجا شروع می شود  
    // ۱. چک کنید آیا p یک Book است؟  
    // و اگر بله، آن را کرده را چاپ کنید.  
    //  
    // ۲. چک کنید آیا p یک Magazine است  
    // و اگر بله، آن را کرده را چاپ کنید.  
}
```

خلاصه بخش ۱۰ و نگاهی به آینده

- تبدیل فرزند به والد. امن، ضمنی و کلید چندریختی. **Upcasting** ✓
- تبدیل والد به فرزند. خطرناک، صریح و برای دسترسی به رفتارهای خاص. **Downcasting** ✓
- تور نجات شما قبل از هر `.Downcast` است. `instanceof` ✓
- خطای زمان اجرا در صورت `Downcast` نامعتبر. `ClassCastException` ✓
- فقط «دید» یا نوع رفرنس را تغییر می‌دهد، نه خود شیء واقعی را. **Casting** ✓

A Look Ahead:

در بخش بعدی، قدرت واقعی چندریختی را با **بازنویسی متدها (Method Overriding)** آزاد خواهیم کرد و می‌بینیم چگونه می‌توان از شر زنجیره‌های `instanceof` خلاص شد.