بازآرایی برنامه
Code Refactoring

صادق علی‌اکبری

سرفصل مطالب

- بازآرایی کد (Code Refactoring) چیست؟
- نیاز به بازآرایی
- مزایای بازآرایی
- تکنیک‌های بازآرایی
- تمرین‌های عملی
  - امکانات محیط‌های توسعه برای بازآرایی

دستخط برنامه‌نویس

- دستخط برنامه‌نویسی شما چطور است؟!

بازآرایی (Refactoring)

- یک فرایند منظم و منضبط برای بازسازی ساختار برنامه
- با هدف بهبود کیفیت کد
- بدون ایجاد تغییر در رفتار برنامه

تعریف بازآرایی

- تغییری در ساختار داخلی نرم‌افزار،
- که باعث می‌شود راحت‌تر خوانده و فهمیده شود،
- و تغییر آن کم‌هزینه‌تر و ساده‌تر شود،
- بدون این که تغییری در رفتار نرم‌افزار مشاهده شود
- مهمترین فایده بازآرایی: افزایش **قابلیت نگهداری** نرم‌افزار

بازآرایی چه نمی‌کند؟ (کارهایی که بازآرایی نیستند)

- تغییر در رفتار برنامه
- ایجاد امکانات جدید
- رفع باگ
  - (معمولاً) بازآرایی زمانی اتفاق می‌افتد که نرم‌افزار به درستی کار می‌کند

- دقت کنید:
  - در حالت عادی وقتی در حال برنامه‌نویسی هستیم، به یکی از کارهای فوق مشغولیم
  - و یا در حال تولید کدهای تست (آزمون واحد) هستیم
- بازآرایی: حالتی جدید در برنامه‌نویسی

بازآرایی چه می‌کند؟

- بهبود ساختار داخلی برنامه
- اجرای فرایندی منظم برای تمیز کردن کد
- بهبود طراحی برنامه بعد از نوشتن کد
  - فرایندهای چابک تولید نرم‌افزار
  - بهبود دایمی طراحی برنامه

بازآرایی چه فوایدی دارد؟

- کاهش احتمال ایجاد شدن باگ
- فراهم آمدن کدهایی که راحت‌تر می‌توانیم با آن‌ها کار کنیم
- افزایش خوانایی برنامه‌ها
  - ساختار داخلی کد شفاف‌تر و صریح‌تر می‌شود
- کاهش پیچیدگی کد
- تسریع برنامه‌نویسی
- کمک به پیدا کردن باگ‌ها
- افزایش قابلیت نگهداری کد
  - امکان تغییر یا گسترش برنامه

فرایند بازآرایی

- در هر مرحله، یک اشکال ساختاری در متن برنامه پیدا می‌کنیم
  - مثلاً یک متد که زیادی طولانی شده است
  - منظور از اشکال، باگ نیست
  - هر یک از این علائم و اشکالات ساختاری، یک «**بوی بد**» در برنامه هستند
  - Bad Smells
- هر «بوی بد»، با یک تکنیک مشخص برطرف می‌شود
  - **تکنیک بازآرایی** (Refactoring Techniques)

مثال

```
Scanner s = new Scanner(System.in);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int a1 = s.nextInt();
System.out.print("Enter the length: ");
int a2 = s.nextInt();

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int b1 = s.nextInt();
System.out.print("Enter the length: ");
int b2 = s.nextInt();

int x = a1*a2;
int y = b1*b2;

if(x == y)
        System.out.println("Equal");
```

- این برنامه را ببینید
- چه اشکالاتی دارد؟
- چگونه ساختار آن را بهبود بخشیم؟

۱- نام‌های نامناسب

```java
Scanner scanner = new Scanner(System.in);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int width1 = scanner.nextInt();
System.out.print("Enter the length: ");
int length1 = scanner.nextInt();

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int width2 = scanner.nextInt();
System.out.print("Enter the length: ");
int length2 = scanner.nextInt();

int area1 = width1*length1;
int area2 = width2*length2;

if(area1 == area2)
System.out.println("Equal");
```

تکنیک: تغییر نام

```
class Rectangle{
	private int length , width;
	public int getLength() {
		return length;
	}
	public void setLength(int length) {
		this.length = length;
	}
	public int getWidth() {
		return width;
	}
	public void setWidth(int width) {
		this.width = width;
	}
	public Rectangle(int length, int width) {
		this.length = length;
		this.width = width;
	}
}
```

تکنیک: استخراج کلاس

```
Scanner scanner = new Scanner(System.in);

System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
int width = scanner.nextInt();
System.out.print("Enter the length: ");
int length = scanner.nextInt();
Rectangle rectangle1 = new Rectangle(length, width);


System.out.println("Rectangle Info.");
System.out.print("Enter the width: ");
width = scanner.nextInt();
System.out.print("Enter the length: ");
length = scanner.nextInt();
Rectangle rectangle2 = new Rectangle(length, width);

int area1 = rectangle1.getWidth()*rectangle1.getLength();
int area2 = rectangle2.getWidth()*rectangle2.getLength();

if(area1 == area2)
        System.out.println("Equal");
```

```
class Rectangle{
      ...
      public int area(){
            return length * width;
      }
}
```

تکنیک: استخراج متد

```
…
int area1 = rectangle1.area();
int area2 = rectangle2.area();
```

```
private static Rectangle readRectangle(Scanner scanner) {
        int width;
        int length;
        System.out.println("Rectangle Info.");
        System.out.print("Enter the width: ");
        width = scanner.nextInt();
        System.out.print("Enter the length: ");
        length = scanner.nextInt();
        Rectangle rectangle2 = new Rectangle(length, width);
        return rectangle2;
}
```

تکنیک: استخراج متد

کد بازآرایی شده:

```
Scanner scanner = new Scanner(System.in);

Rectangle rectangle1 = readRectangle(scanner);
Rectangle rectangle2 = readRectangle(scanner);

int area1 = rectangle1.area();
int area2 = rectangle2.area();

if(area1 == area2)
    System.out.println("Equal");
```

مرور مثال

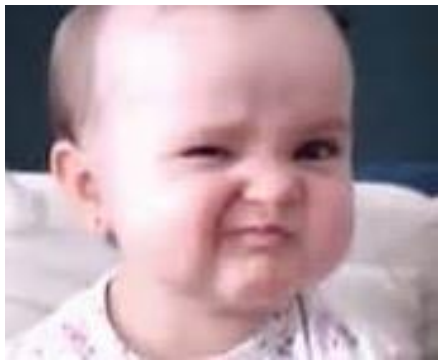
- کدی که به درستی کار می‌کرد
- ساختار داخلی کد بهبود یافت
- در هر مرحله، یک «**بوی بد**» در متن برنامه پیدا کردیم
  - مثلاً نامگذاری نامناسب، کد تکراری، و ...
- هر بوی بد را با کمک یک تکنیک بازآرایی رفع کردیم

- **بازآرایی**=پیدا کردن بوی بد + رفع آن با کمک تکنیک متناسب بازآرایی

بوهای بد در کد و تکنیک‌های بازآرایی

«بوی بد» در برنامه

- هر علامتی که ممکن است نشان از یک مشکل عمیق‌تر در برنامه باشد
- خطایی در ساختار برنامه که (فعلاً) ایجاد اشکال نمی‌کند
  - ولی در درازمدت مشکل‌ساز خواهد شد (ایجاد باگ، دشواری تغییر و غیره)
- بوی بد، باگ نیست
  - ولی روند توسعه نرم‌افزار را کند، سخت، پرهزینه و خطاخیز می‌کند



- این اصطلاح توسط Kent Beck رایج شد
- «اگه بو میده، عوضش کن!»
  - *If it stinks, change it!*
- بوی بد کد توسط تکنیک‌های مشخص بازآرایی قابل رفع هستند

افراد مهم در حوزه بازآرایی

- Kent Beck
  - از پیشروان Extreme Programming
  - از صاحب‌نظران موضوع آزمون واحد و از طراحان JUnit

- Martin Fowler
  - صاحب نظر در زمینه تحلیل و طراحی شیءگرا
  - سایت خوب http://martinfowler.com/

مثال

- بوی بد «نام نامناسب»
  - متغیر، کلاس یا متدی که نام گویا و مناسبی ندارد
  - فهم برنامه را سخت می‌کند
- تکنیک بازآرایی: تغییر نام (Rename)
- این مثال را در عمل تمرین کنیم

بوی بد: کد تکراری (Duplicated Code)

- قطعه‌کدی **یکسان** و یا **بسیار مشابه** که بیش از یک جا دیده شود
- قطعاً یک علامت بد است
  - تغییر در منطق این بخش، مستلزم تغییر همه تکرارهای آن است
  - رفع اشکال یکی، باید در همه انجام شود
- در زمان برنامه‌نویسی، از «copy/paste» پرهیز کنید
- تکنیک‌های بازآرایی
  - استخراج متد (Extract Method)
  - استخراج متغیر (Extract Variable)
  - استخراج کلاس (Extract Class)

بوی بد: متد طولانی (Long Method)

- متدهای طولانی به سختی فهمیده می‌شوند
- تغییر آن‌ها سخت‌تر است
- یک متد با چند خط طولانی است؟
  - قانون مشخصی در این زمینه وجود ندارد
- یک متد خوب، «کاری منسجم و مستقل» انجام می‌دهد، نه چند کار مختلف
- تکنیک بازآرایی: (معمولاً) استخراج متد

بوهای بد در کد

- کلاس طولانی (Large Class)
  - تکنیک: استخراج کلاس، استخراج subclass یا superclass
- تعداد زیاد پارامترهای یک متد (too many parameters)
  - تکنیک: فراخوانی متد به جای پاس شدن مقدار پارامتر (Replace Parameter with Method Call)
  - تکنیک: تبدیل مجموعه پارامترها به یک شیء (Introduce Parameter Object)
- حسادت به داشته‌های دیگران (Feature Envy)
  - متد/متدهایی که از بیشتر از طرف یک کلاس دیگر فراخوانی می‌شود
  - کلاسی که متدهای کلاسی دیگر را بیش از حد فراخوانی می‌کند
  - تکنیک: انتقال متد (Move Method)

مطالعه تکمیلی: سایر بوهای بد در کد

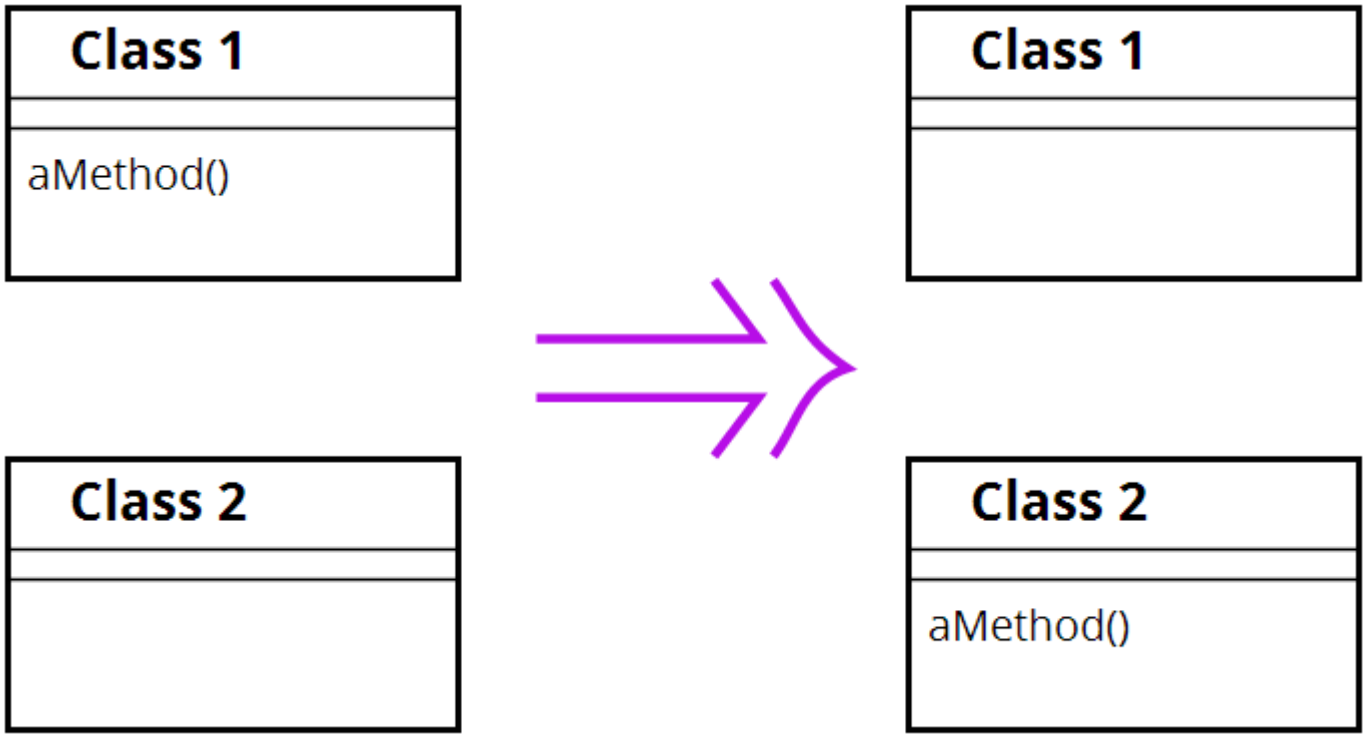
- Alternative Classes with Different Interfaces
- Incomplete Library Class
- Data Class
- Refused Bequest
- Comments
- Metaprogramming Madness
- Disjointed API
- Repetitive Boilerplate
- Primitive Obsession
- Case Statements
- Parallel Inheritance Hierarchies
- Speculative Generality
- Temporary Field
- Message Chains
- Middle Man
- Inappropriate Intimacy

تکنیک‌های بازآرایی

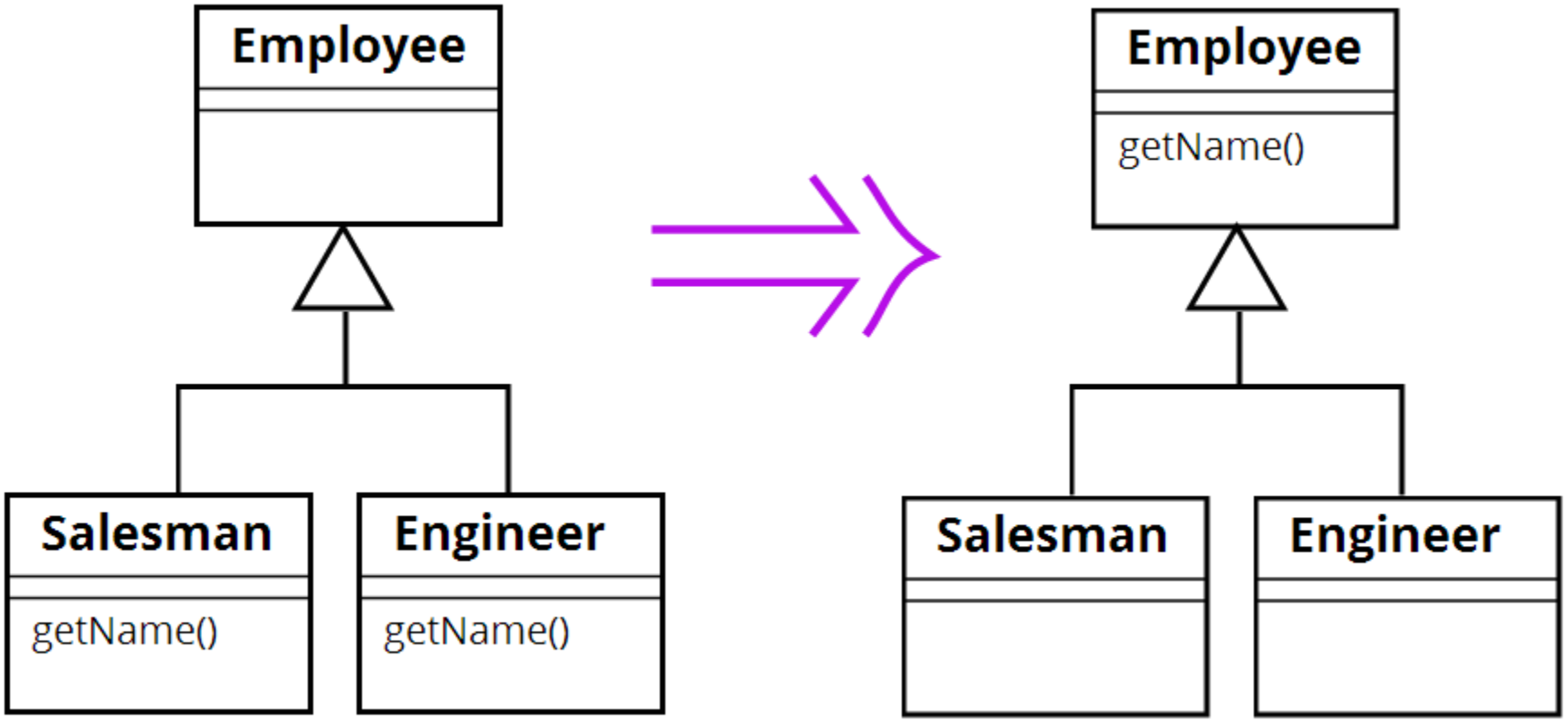
- تغییر نام (rename)
  - کلاس، متد، متغیر
- استخراج متد (extract method)
- متد درخط (inline)
- استخراج کلاس (extract class)
- کلاس درخط

تکنیک انتقال (Move)

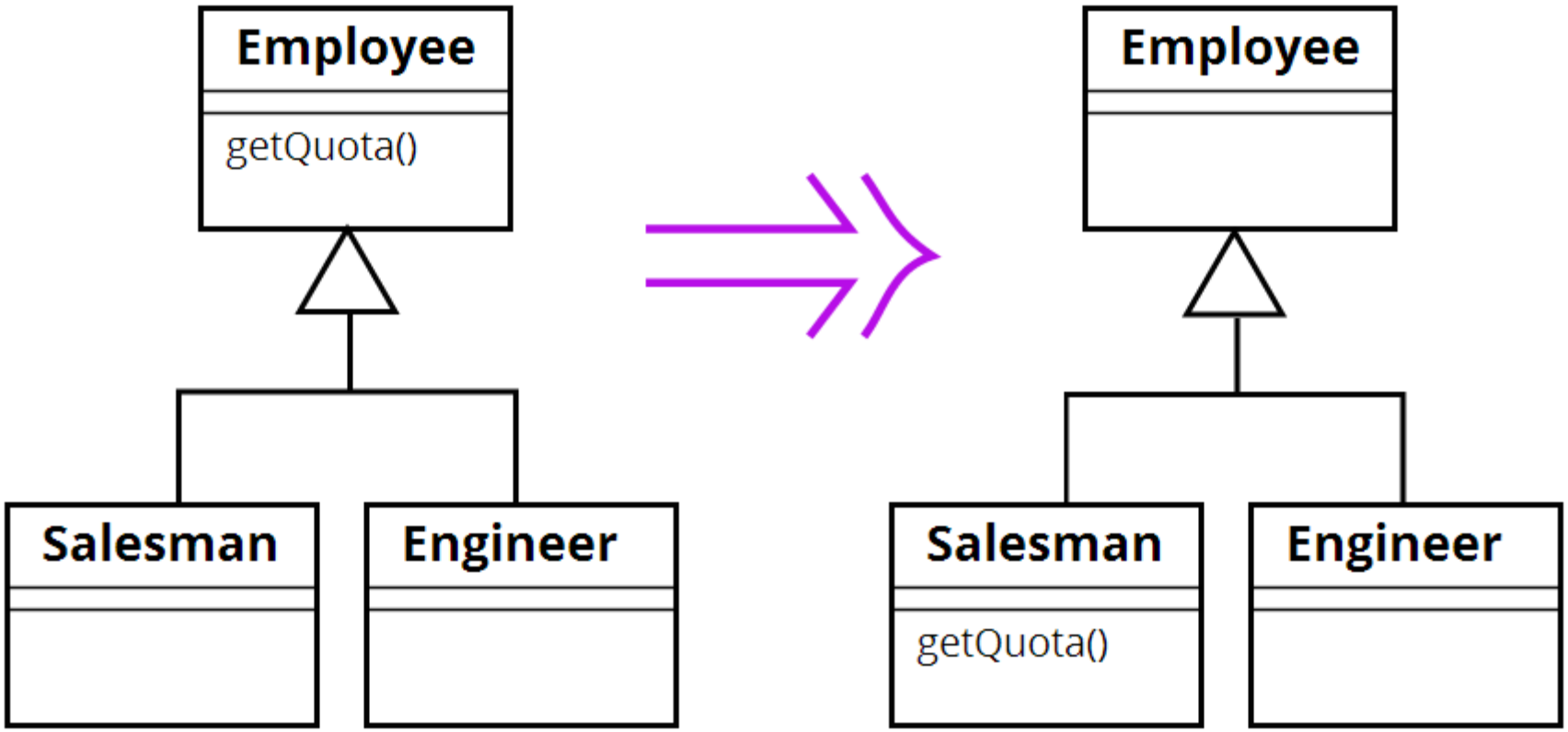
- انتقال کلاس، متد، متغیر
  - به یک کلاس یا بسته (package) دیگر
  - مثال:



بالا کشیدن متد (Pull up method)



پایین آوردن متد (Push Down method)

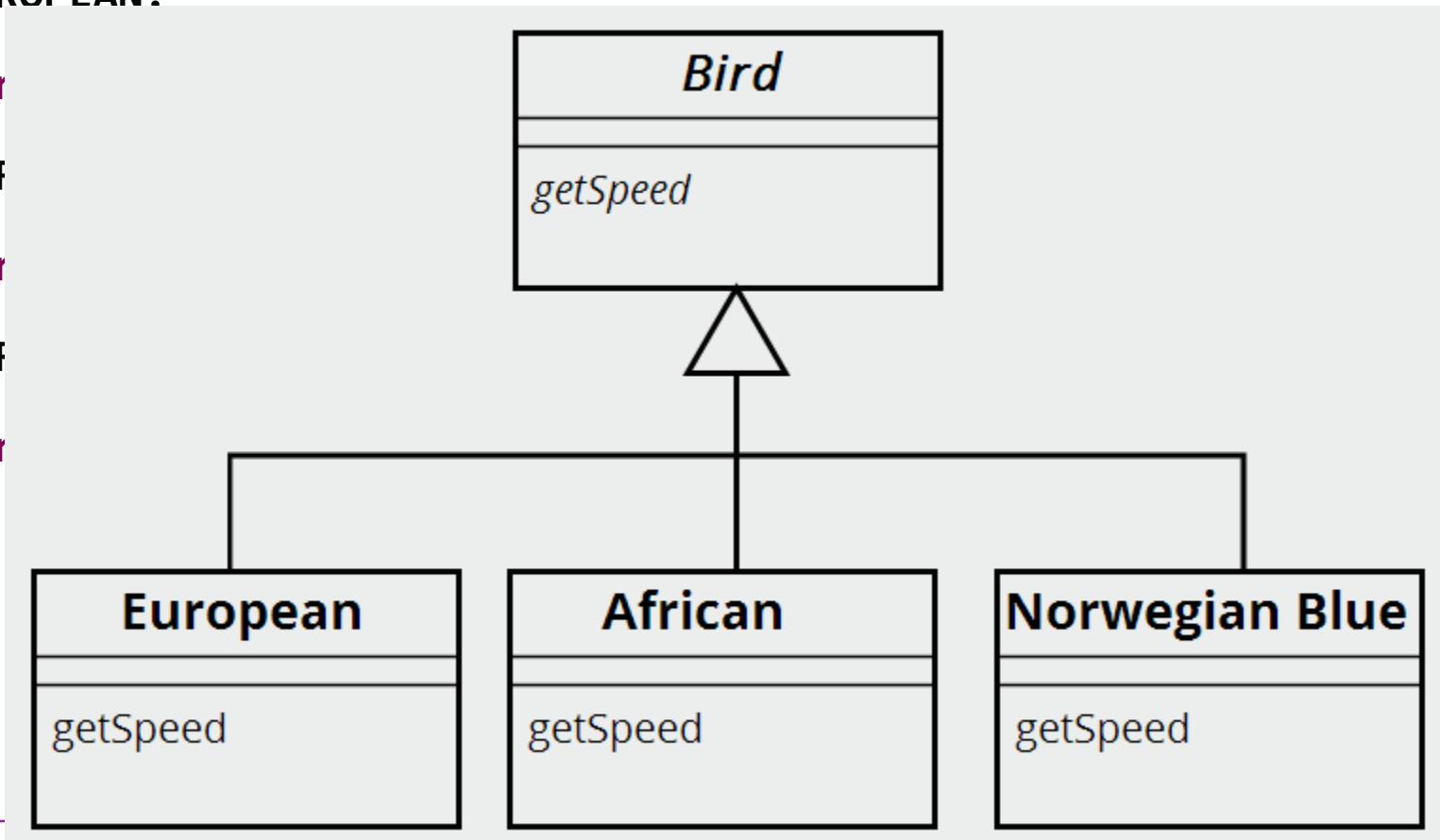


تکنیک‌های بازآرایی

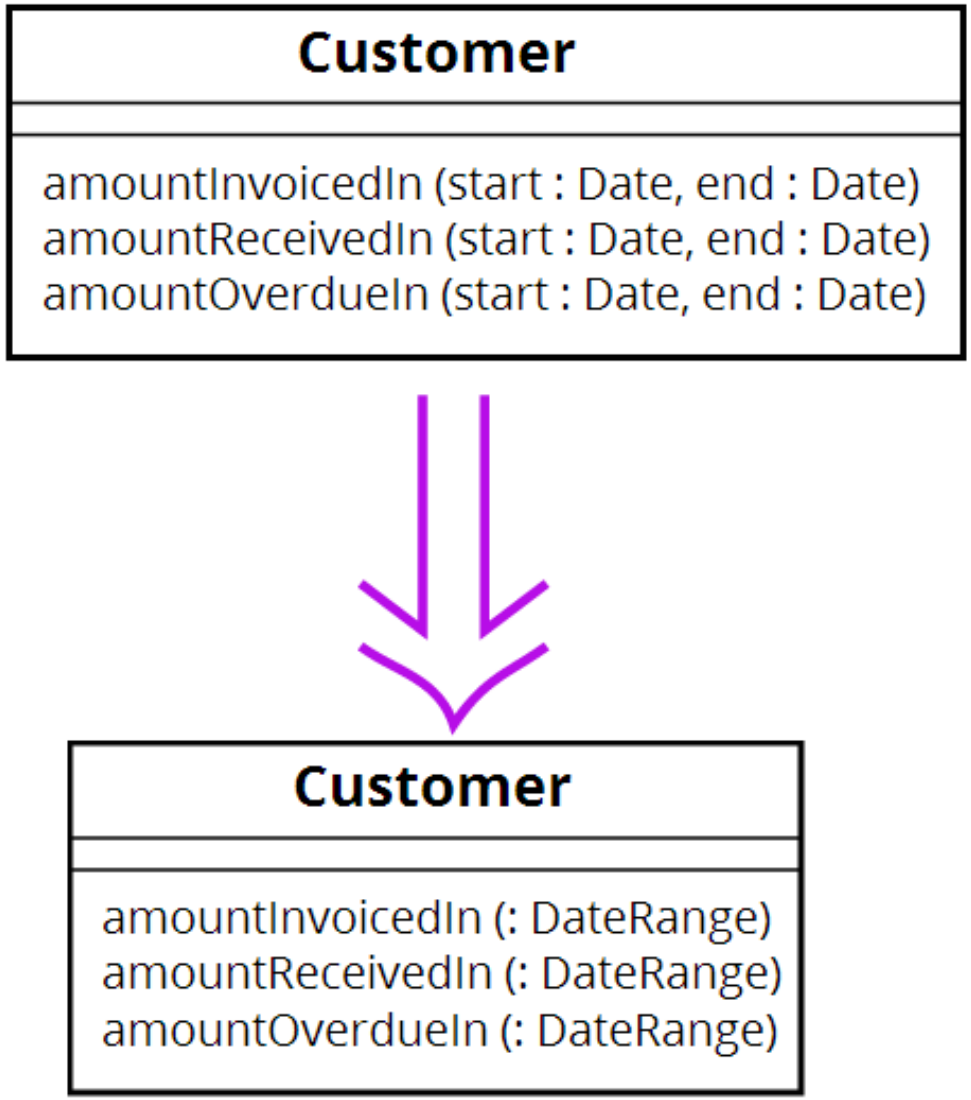
- استخراج مقدار ثابت (Extract Constant)
  - مثلاً pi=3.14;
  - اعداد جادویی (magic numbers)
    - به اعداد ثابت در برنامه‌های خود نام بدهید
- تغییر امضای متد (Change Method Signature)
  - کم و زیاد کردن پارامتر
  - تغییر نوع برگشتی
  - تغییر سطح دسترسی
  - تغییر خطاهای پرتابی (Exceptions)

تبدیل شروط (If) به چندریختی (Polymorphism)

```
double getSpeed() {
  switch (_type) {
    case EUROPEAN:
      return
    case AFR
      return
    case NOR
      return
  }
}
```



معرفی شیء پارامتر (Introduce Parameter Object)



- وقتی که تعدادی پارامتر معمولاً همراه هم پاس می‌شوند

سایر تکنیک‌های بازآرایی

- نگاهی به این‌جا بیاندازید:
- http://refactoring.com/catalog/
- کاتالوگی از تکنیک‌های بازآرایی
- با تشکر از مارتین فاولر عزیز

استعاره دو کلاه

- در هنگام برنامه‌نویسی، زمان خود را به دو بخش **مجزا** تقسیم کنید:
  - تولید برنامه
  - بازآرایی

- در هر مورد، **کلاه** مخصوص همان نقش را روی سرتان قرار دهید
- در هنگام تولید برنامه، درگیر بازآرایی نشوید
- در هنگام بازآرایی، امکانات جدید ایجاد نکنید
- شاید به کرّات و به سرعت، بین این دو حالت نقش عوض کنید
  - اما هر نقش باید به طور مستقل ایفا شود
  - در هنگام ایفای یک نقش، نقش دیگر را بازی نکنید

پشتیبانی از بازآرایی در محیطهای توسعه

- محیطهای یکپارچه توسعه (IDE) امکاناتی برای بازآرایی ارائه می‌کنند
- Eclipse, IDEA, NetBeans, ...
- اجرای تکنیک‌های بازآرایی را خودکار می‌کنند
- اشتباهات انسانی را کاهش می‌دهند
- و اجرای تکنیک‌ها را تسریع می‌بخشند
- البته دانش، و مهارت بازآرایی هم بسیار مهم است
  - بازآرایی یک فرایند کاملاً خودکار نخواهد بود
  - انتخاب اشکال (بوی بد)، تکنیک بازآرایی و نحوه اجرای تکنیک: بر عهده برنامه‌نویس
  - ابزارها فقط کمک می‌کنند

تمرین عملی باز آرایی در یک محیط توسعه

تمرین عملی

- تمرین تکنیک‌های بازآرایی
- استفاده از استعاره دو کلاه
- تبیین جایگاه آزمون واحد در بازآرایی

مطالب تکمیلی

زمان بازآرایی

- وقتی امکانات جدیدی به برنامه اضافه می‌کنید
- وقتی یک باگ را برطرف می‌کنید
- همین طور که مرور کد (code review) می‌کنید
  - و البته وقتی که ابزارهای تحلیل کد، اشکالاتی را گزارش می‌کنند

تأثیر بازآرایی در کارایی (PERFORMANCE)

- برخی به بازآرایی انتقاد می‌کنند که:
  - تکنیک‌های بازآرایی باعث می‌شود کارایی برنامه کاهش پیدا کند
  - مثلاً تعداد متدها و فراخوانی متدها بیشتر می‌شود
  - یا تعداد متغیرها و فضای حافظه اشغالی بیشتر می‌شود
- در واقع برخی از تکنیک‌های بازآرایی کارایی را افزایش هم می‌دهند
- تأثیر بقیه تکنیک‌ها هم در کارایی معمولاً ناچیز است
- فایده بازآرایی: ساختار کد **قابل بهبود** می‌شود (مثلاً از نظر کارایی)
- توصیه مهم: ابتدا نرم‌افزاری قابل بهبود بنویسید،
  - سپس در صورت لزوم آن را برای رسیدن به کارایی بهتر بهبود بخشید
- Write tunable software first and then tune it for sufficient speed

ریسک بازآرایی

- بازآرایی، ذاتاً مخاطره‌آمیز (Risky) است
- زیرا برنامه‌ای را تغییر می‌دهد که کار می‌کند
  - ممکن است بازآرایی به ایجاد باگ‌های جدید منجر شود
- چطور مدیر را برای چنین کاری متقاعد کنیم؟
- پیشنهاد مارتین فاولر:
  - اگر مدیر شما یک فرد فنی نیست،
  - لازم نیست به مدیر بگویید یا اجازه بگیرید!
- بازآرایی، بخشی از کار شماست و در تخمین زمان فرض می‌شود
- زمانی که صرف بازآرایی شده، تولید آینده شما را تسریع می‌کند

- البته مدیرها از روی کلاهی که به سر دارید، می‌فهمند! (-;

مهار خطر بازآرایی

- انجام بازآرایی به صورت سیستماتیک
- استفاده از ابزارها و امکانات IDE
- انجام قدم‌های کوچک
- استفاده از تست
  - موهبت تست در کنترل دایمی کیفیت
- ترسو نباشید: «شجاعت» هم لازم است
  - پنج ارزش در XP :

1. تعامل (ارتباطات)
2. سادگی (در آغاز)
3. بازخورد (فیدبک از سیستم، از مشتری، از تیم)
4. **شجاعت**
5. احترام

جایگاه بازآرایی در متدولوژی‌های چابک

- بخشی مهم از متدولوژی‌های چابک
  - مثل XP (Extreme Programming)
- در کنار موضوعات دیگری مانند
  - آزمون واحد
  - مرور کد
  - برنامه‌نویسی دونفری
- متدولوژی‌های چابک، تغییر را می‌پذیرند
  - تغییر در طراحی، تغییر در نیازمندی، تغییر در ساختار کد و ...
- بازآرایی بخشی جدانشدنی از متدولوژی‌های چابک است

مخالفان بازآرایی

- دلایلی که بر ضد بازآرایی می‌آورند
  - وقت نداریم، پروژه از زمانبندی عقب است!
  - زمان زیادی برای بازآرایی هدر می رود
  - بازآرایی، کار و وظیفه من نیست
- ناله‌هایی آشنا که گاهی در مخالفت با دیگر بایدها نیز گفته می‌شود
  - به خصوص درباره تست و آزمون واحد
- عدم انجام بازآرایی: وام فنی

بازآرایی برای انطباق با الگوهای طراحی

- تفاوت «بوی بد» و پادالگو (anti pattern)
  - الگوهای طراحی مفاهیم سطح بالاتری هستند
  - معمولاً: عدم تشخیص پادالگوها توسط ابزارهای خودکار (مثل Sonar)
  - معمولاً: عدم پشتیبانی از بازآرایی پادالگوها توسط محیط‌های توسعه (مثل Eclipse)
  - معمولاً: در تشخیص محل استفاده از الگوی نرم‌افزاری، تجربه و مهارت بیشتری لازم است
  - گاهی هم این مفاهیم همپوشانی دارند: Large Class و God Object
- معنای refactoring to patterns
- بازآرایی به منظور رعایت الگوهای طراحی
  - گاهی: سرعت برنامه‌نویسی و سرعت تغییرات ← عدم رعایت الگوهای طراحی
  - وگاهی به دلیل پرهیز از over-engineering
  - نیاز به بازنویسی و بازآرایی
  - کتاب Refactoring to Patterns

دردسرهای بازآرایی

- گاهی بازآرایی **مشکلاتی** ایجاد می‌کند و یا **هزینه** زیادی دارد
  - در این موارد، باید به دقت ابعاد و تبعات بازآرایی بررسی شود

1. تغییر در واسطهای منتشر شده (published interfaces)
   - واسطهایی که دیگران در حال استفاده از آن‌ها هستند
   - گاهی مجبور می‌شویم یک نسخه قدیمی از واسط را حفظ کنیم
     - به صورت deprecated
2. تغییر در طراحی پایگاه داده (schema)
   - بسیاری از برنامه‌ها به شدت به طراحی پایگاه داده وابسته هستند
   - مهاجرت داده‌های موجود: مشکلی دیگر
3. بازآرایی تصمیمات اساسی معماری و طراحی
   - مثل انتخاب تکنولوژی

تحلیل استاتیک کد

- ابزارهایی برای تحلیل ساختار برنامه (متن برنامه‌ها) وجود دارند
  - بسیاری از «بوهای بد» را به صورت خودکار کشف می‌کنند
- ابزارهایی مانند
  - Checkstyle, PMD, FindBugs
  - و البته SonarQube

افراد مهم در حوزه بازآرایی

- Kent Beck
  - از پیشروان Extreme Programming
  - از صاحب‌نظران موضوع آزمون واحد و از طراحان JUnit

- Martin Fowler
  - صاحب نظر در زمینه تحلیل و طراحی شیءگرا
  - سایت خوب http://martinfowler.com/

جمع‌بندی

جمع‌بندی

- نیاز به بازآرایی
- فواید بازآرایی
- بوی بد در برنامه
- تکنیک‌های بازآرایی
- ابزارهای خودکار برای بازآرایی
  - امکانات IDE
  - ابزارهای کشف بوی بد

نقل قول

- If it smells bad, change it!

  (Grandma Beck, discussing child-rearing philosophy)

- Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

  (Martin Fowler)

کتاب‌های مرجع

- **Refactoring**: Improving the Design of Existing Code
  - Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts
    - کتابی قدیمی، ولی همچنان زنده و پرخواننده (همین کتاب را پیشنهاد می‌کنیم)
- **Refactoring**, Ruby Edition
  - Jay Fields, Shane Harvie, Martin Fowler, and Kent Black
    - بسیار شبیه نسخه قبلی همین کتاب
    - البته برنامه‌ها به جای جاوا، این بار با Ruby نوشته شده‌اند

برخی صفحات مفید

- http://en.wikipedia.org/wiki/Code_refactoring
- http://refactoring.com/
- http://refactoring.com/catalog/
- http://sourcemaking.com/refactoring

تمرین

- کدهای خودتان را بازآرایی کنید
- تکنیک‌های بازآرایی را در محیط توسعه مورد علاقه‌تان بررسی کنید
- کدهایی که همراه این ویدیو ارائه می‌شود را بازآرایی کنید

لطیفه

- وقتی به مدیر پروژه بگویید: می‌خواهم برنامه‌هایم را بازآرایی کنم
  - برنامه‌هایی که کار می‌کنند
  - و ممکن است این کار حتی باعث به وجود آمدن اشکالاتی در برنامه شود
- چهره مدیر پروژه چگونه خواهد بود؟




پایان

سایر مطالب

Martin Fowler:

- The base books and workbooks look at the fundamentals of refactoring software. They are where to start, but refactoring applies in more places and contexts, and several books explore these further aspects.
- There is a close relationship between refactoring and **patterns**. Often the best way to use patterns is to gradually refactor your code to use the pattern once you realize it's needed. Joshua Kerievsky's Refactoring to Patterns explores this topic, making this a great topic to learn about once you've got the basic refactorings under your belt.