درس «مهندسی نرم‌افزار ۲»

بازآرایی برنامه
Code Refactoring

فائزه گوهری

سرفصل مطالب

- بازآرایی کد (Code Refactoring) چیست؟
- نیاز به بازآرایی
- مزایای بازآرایی
- بوهای بد در کد و تکنیک‌های بازآرایی

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 (نگهداری) آن کم هزینه‌تر 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
  - بخصوص در فرایندهای چابک تولید نرم‌افزار
  - بهبود دائمی طراحی 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
  - تکنیک‌های 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؟

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");
```

۱- اسامی نامناسب برای متغیرها

مثال

```
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

مثال

```
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");
```

۲- دسته داده‌ها

(تکرار **گروهی** از متغیرها

در نقاط مختلف کد)

مثال

- تعریف کلاس مستطیل با دو متغیر طول و عرض

```
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

مثال

```
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");
```

- بازآرایی کد اولیه بر اساس کلاس شناسایی شده جدید (کلاس مستطیل)
- قابلیت استفاده مجدد از کلاس مستطیل به تعداد دلخواه

مثال

```
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");
```

۳- قطعه کد تکراری

مثال

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

تکنیک استخراج متد

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

مثال

```
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);
```

۴- قطعه کد تکراری

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

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

مثال

```
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 rectangle = new Rectangle(length, width);
        return rectangle;
}
```

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");
```

مقایسه کد اولیه با کد بازآرایی شده

```
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");
```

```
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");
```

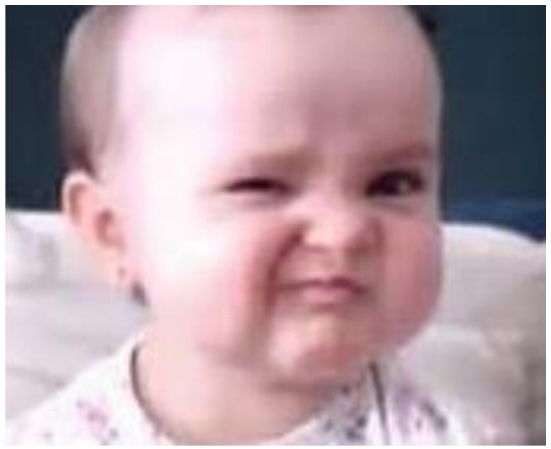
مرور مثال

- 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 تکنیک های 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 و نگهداری 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



بوی بد: کد تکراری (DUPLICATED CODE)

- قطعه کدی **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 برنامه نویسی، از «copy/paste» پرهیز کنید
- تکنیک های بازآرایی
  - استخراج متد (Extract Method)
  - استخراج متغیر (Extract Variable)
  - استخراج کلاس (Extract Class)

```
public void m() {
    double b= 5;
    double c=7;
    if (b*c < 8) {
        double d= b*c;
    }
    b= b*c;
}
```

مثلا برای جلوگیری از تکرار b*c در این کد، باید از تکنیک استخراج متغیر استفاده کنیم

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 (به زبان برنامه‌نویسی بسیار وابسته است)
  - یک قاعده سرانگشتی این است که تعداد خطوط یک متد باید در حدی باشد که بدون نیاز به اسکرول بر روی اسکرین قابل مشاهده باشد (مثلا بین ۵ تا ۱۵ خط)
- یک متد خوب، «کاری منسجم و مستقل» انجام می‌دهد،

  نه چندین کار مختلف و غیر مرتبط
  - انسجام (high cohesion)
  - استقلال (low coupling)
- تکنیک بازآرایی: (معمولاً) استخراج متد

بوی بد: دسته داده‌ها (DATA CLUMPS)

- گروهی از داده‌ها که معمولاً **با هم** مورد استفاده قرار می‌گیرند
- مثلاً: نام، نام خانوادگی، شناسه و گذرواژه
- در یک کد ضعیف، این گروه از داده‌ها در نقاط مختلف کد، تکرار می‌شوند (معمولا بدلیل عادت copy/paste)
  - تکنیک بازآرایی:

1. استخراج کلاس: اگر داده‌های تکراری در داخل بدنه یک کلاس یا متد استفاده شوند (مشابه مثالی که قبلا درباره مستطیل دیدیم)
2. معرفی شی پارامتر (Introduce Parameter Object): اگر داده‌های تکراری بعنوان پارامترهای یک متد پاس داده شوند.

مثال از تکنیک بازآرایی «معرفی شی پارامتر» جهت رفع دسته داده‌ها

- سه متغیر مربوط به مختصات x، y و z معمولا بصورت گروهی با هم استفاده می‌شوند
  - در این مثال: بعنوان پارامترهای ورودی متد AddCoords

Prior to refactor

```
public void AddCoords(int x, int y, int z) { /* … */ }
```

مثال از تکنیک بازآرایی «معرفی شی پارامتر» جهت رفع دسته داده‌ها

Post refactor

```
public void AddCoords(Coords coords) { /* … */ }

public class Coords
{
    public Coords(int x, int y, int z)
    {
        X = x;
        Y = y;
        Z = z;
    }

    public int X { get; }
    public int Y { get; }
    public int Z { get; }
}
```

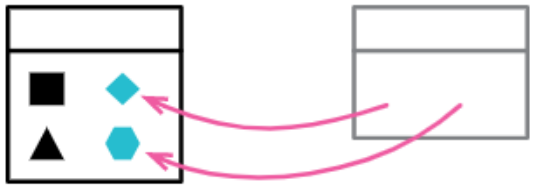
- بازآرایی کد با تکنیک معرفی شی پارامتر
  - قرار دادن هر سه پارامتر ورودی در یک کلاس

بوی بد: کلاس بزرگ (LARGE CLASS)

- کلاس بزرگ و طولانی
- کلاسی که حاوی داده‌ها، متدها و خطوط کد زیاد باشد.
- یک کلاس خوب، باید «منسجم و مستقل» باشد.
- تکنیک بازآرایی: استخراج کلاس، استخراج subclass یا superclass

بوی بد: کلاس تنبل (LAZY CLASS)

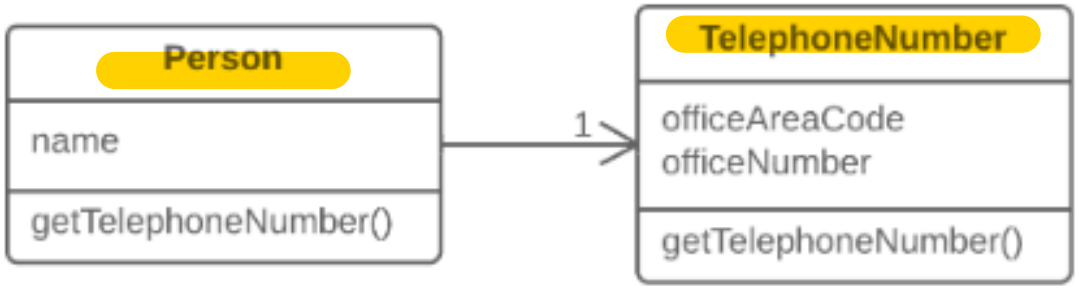
- کلاسی که بسیار مختصر است
  - مثلا فقط یک متد دارد و به ندرت توسط دیگر کلاس‌ها استفاده می‌شود
- کوچک تر از آن است که یک کلاس مستقل باشد
- تکنیک بازآرایی: کلاس درخط (Inline Class)
  - انتقال تمام فیچرهای یک کلاس به کلاس دیگر



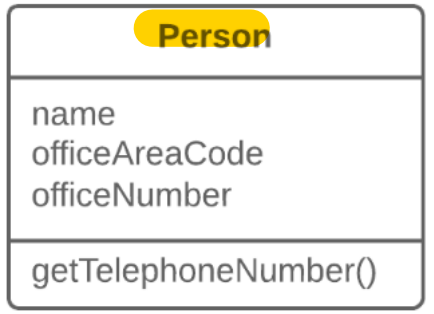
مثال از تکنیک بازآرایی «کلاس درخط»

- کلاس تنبل در این کد: TelephoneNumber

```
class Person {
  get officeAreaCode() {return this._telephoneNumber.areaCode;}
  get officeNumber()   {return this._telephoneNumber.number;}
}
class TelephoneNumber {
  get areaCode() {return this._areaCode;}
  get number()   {return this._number;}
}
```



```
class Person {
  get officeAreaCode() {return this._officeAreaCode;}
  get officeNumber()   {return this._officeNumber;}
```



بوی بد: تعداد زیاد پارامترهای یک متد (Long Parameter List)

- مثلا بیش از ۳ یا ۴ پارامتر برای یک متد
- تکنیک: تبدیل مجموعه پارامترها به یک شیء (Introduce Parameter Object)
- تکنیک: فراخوانی متد به جای پاس شدن مقدار پارامتر

(Replace Parameter with Method Call)

Problem: Calling a query method and passing its results as the parameters of another method, while that method could call the query directly.

```
int basePrice = quantity * itemPrice;
double seasonDiscount = this.getSeasonalDiscount();
double fees = this.getFees();
double finalPrice =
      discountedPrice(basePrice, seasonDiscount, fees);
```

Solution: Instead of passing the value through a parameter, try placing a query call inside the method body.

```
int basePrice = quantity * itemPrice;
double finalPrice =
            discountedPrice(basePrice);
```

بوی بد: حسادت به داشته های دیگران (Feature Envy)

- متد/متدهایی که 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

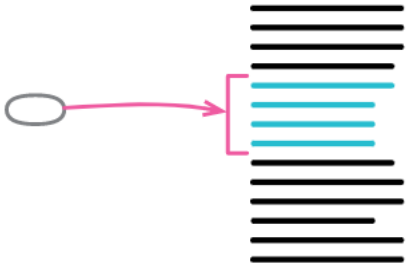
- Alternative Classes with Different Interfaces
- Incomplete Library Class
- Data Class
- Refused Bequest
- Comments
- Metaprogramming Madness
- Disjointed API
- Repetitive Boilerplate
- Primitive Obsession
- Switch Statements
- Parallel Inheritance Hierarchies
- Speculative Generality
- Temporary Field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

- برای مطالعه بیشتر به این آدرس رجوع کنید:

https://refactoring.guru/refactoring/smells

تکنیک‌های بازآرایی

- تغییر نام (Rename)
  - کلاس، متد، متغیر
- استخراج متد (Extract Method)
- استخراج کلاس (Extract Class)
- کلاس درخط (Inline Class)
  - برعکس تکنیک استخراج کلاس
- متد درخط (Inline Method)
  - جایگزین کردن فراخوانی یک متد با بدنه آن متد و حذف متد مربوطه
  - برعکس تکنیک استخراج متد



مثال از تکنیک بازآرایی «متد درخط»

- مثال: امتیازدهی به یک راننده پیک تحویل غذا بر اساس تعداد دفعات تاخیر در تحویل

```
function getRating(driver) {
  return moreThanFiveLateDeliveries(driver) ? 2 : 1;
}

function moreThanFiveLateDeliveries(driver) {
  return driver.numberOfLateDeliveries > 5;
}
```

متد ...moreThan را صدا می‌زند. اگر نتیجه true باشد، امتیاز ۲ و در غیر اینصورت مقدار ۱ را بر می‌گرداند

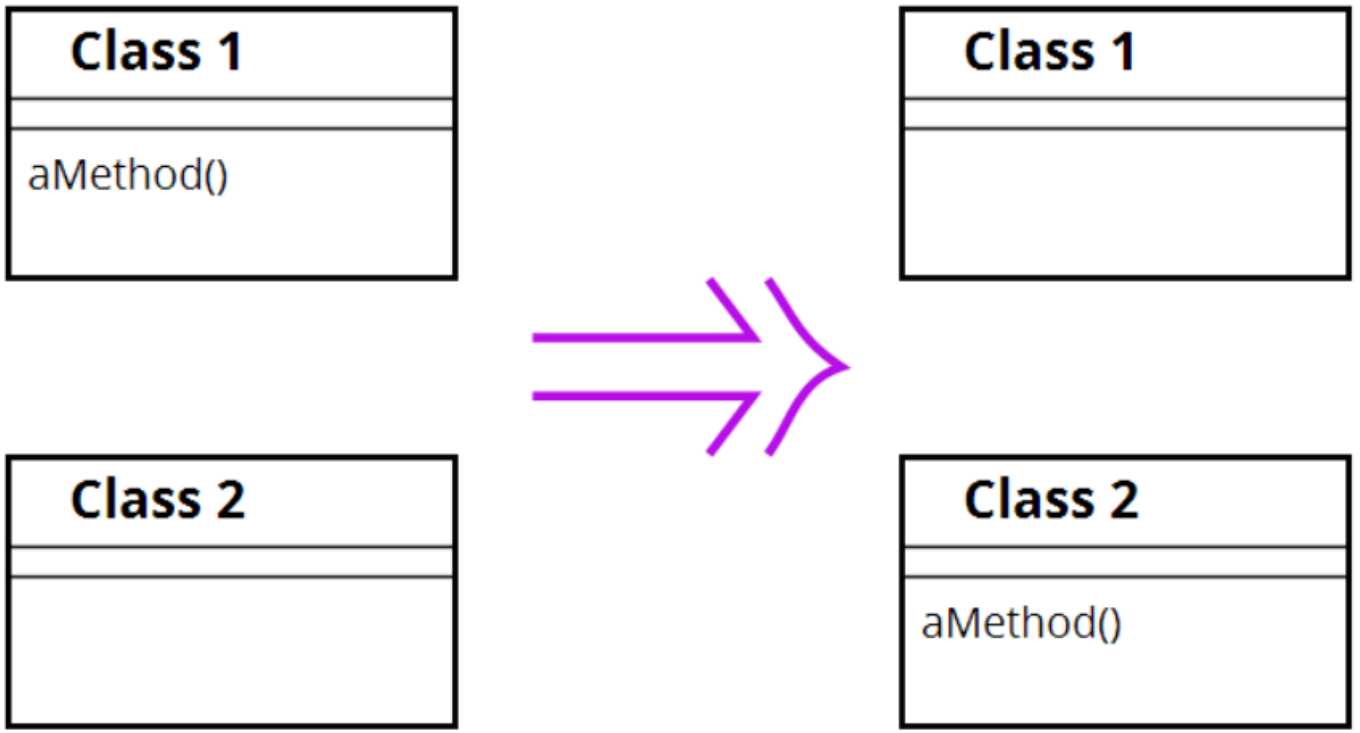
اگر تعداد دفعات تاخیر بیش از ۵ مرتبه باشد، true و در غیر اینصورت false بر می‌گرداند

```
function getRating(driver) {
  return (driver.numberOfLateDeliveries > 5) ? 2 : 1;
}
```

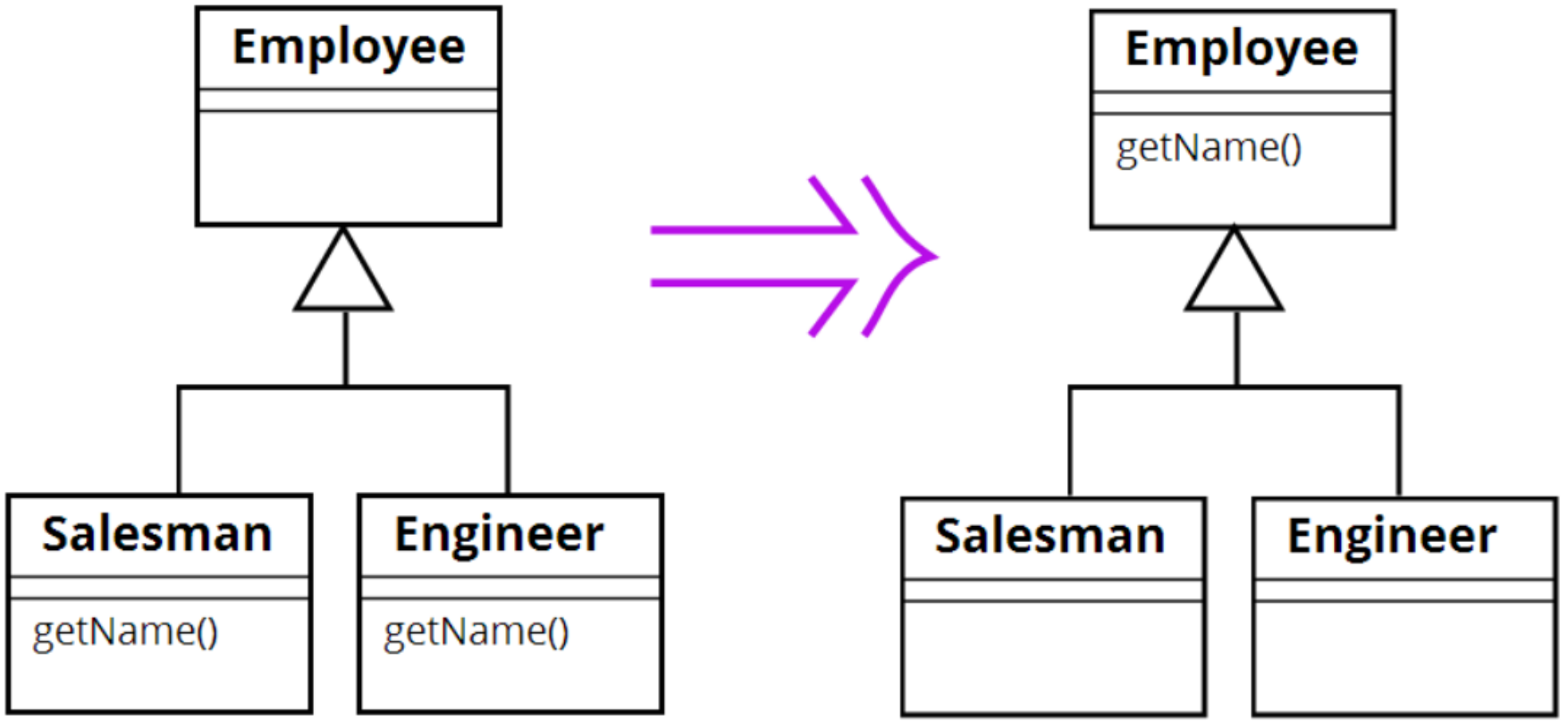
حذف متد ...moreThan و جایگزینی بدنه آن به جای فراخوانی متد

تکنیک انتقال (MOVE)

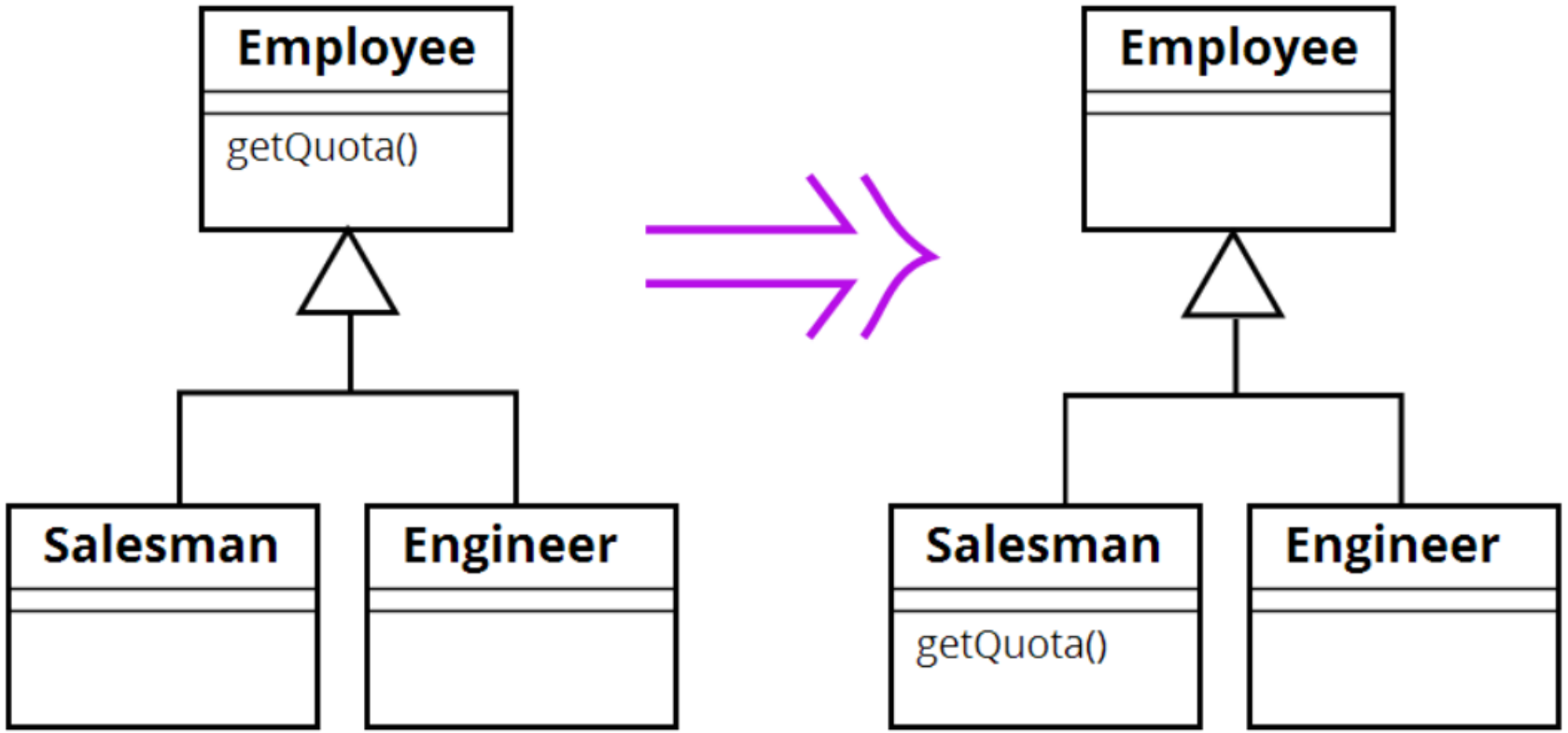
- انتقال کلاس، متد، متغیر
  - به یک کلاس یا بسته (package) دیگر
  - مثال:



تکنیک بالا کشیدن متد (PULL UP METHOD)



تکنیک پایین آوردن متد (PULL DOWN METHOD)



تکنیک‌های بازآرایی

```
function potentialEnergy(mass, height) {
  return mass * 9.81 * height;
}
```

```
const STANDARD_GRAVITY = 9.81;
function potentialEnergy(mass, height) {
  return mass * STANDARD_GRAVITY * height;
}
```

- 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 SWITCH به چندریختی (POLYMORPHISM)

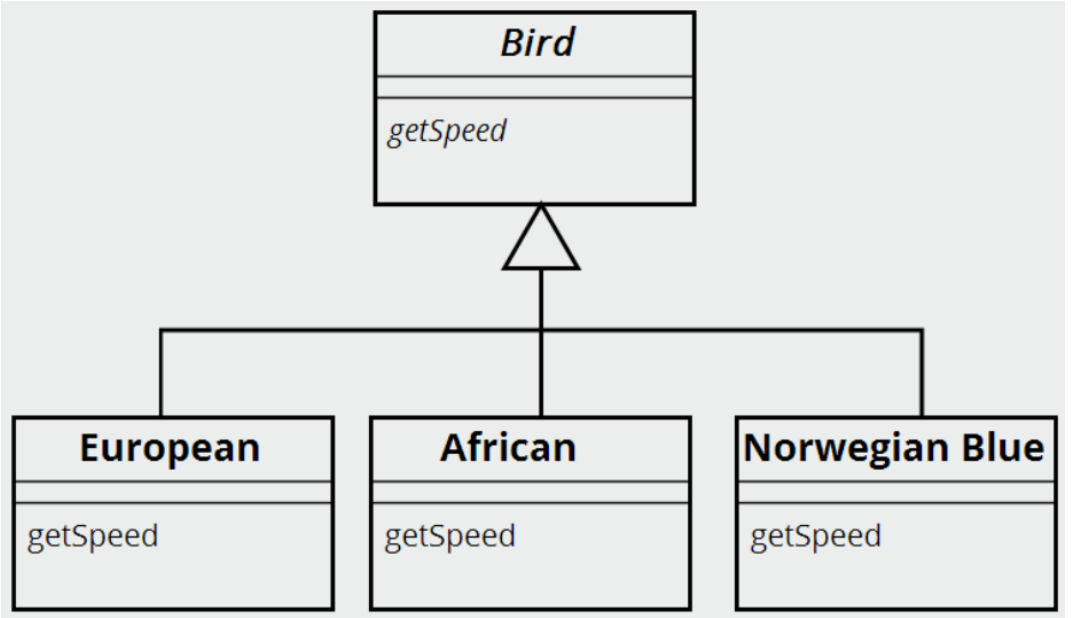
- جملات پیچیده switch یا جملات تو در تو if
- قبل از بازآرایی:

```
class Bird {
  // ...
  double getSpeed() {
    switch (type) {
      case EUROPEAN:
        return getBaseSpeed();
      case AFRICAN:
        return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;
      case NORWEGIAN_BLUE:
        return (isNailed) ? 0 : getBaseSpeed(voltage);
    }
    throw new RuntimeException("Should be unreachable");
  }
}
```

تبدیل SWITCH به چندریختی (POLYMORPHISM)

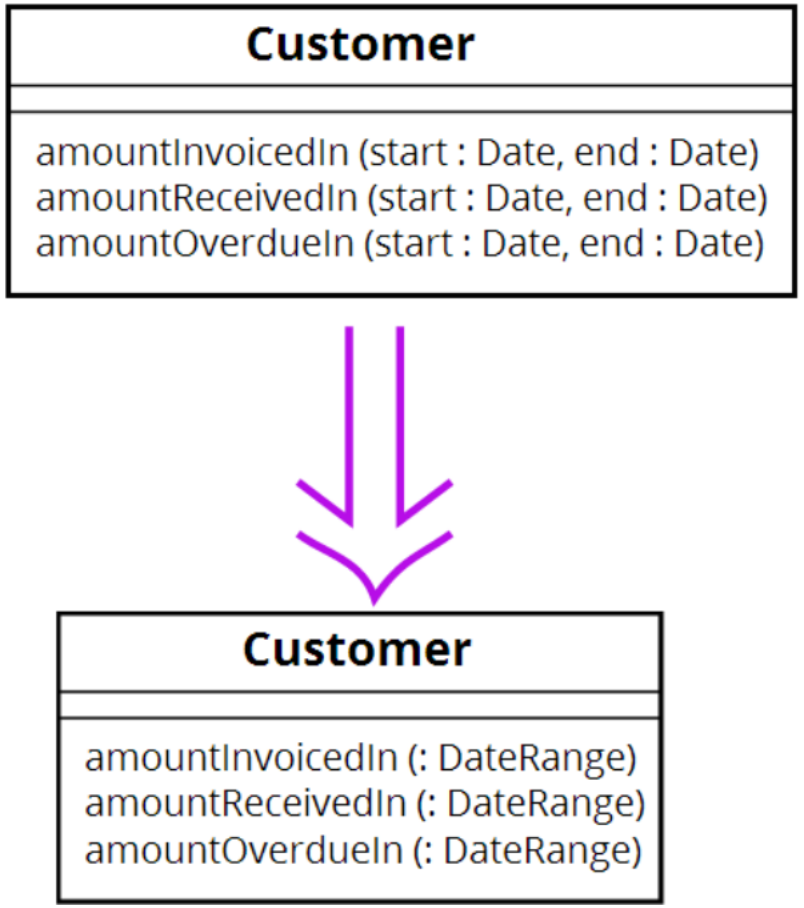
```
abstract class Bird {
  // ...
  abstract double getSpeed();
}

class European extends Bird {
  double getSpeed() {
    return getBaseSpeed();  }
}
class African extends Bird {
  double getSpeed() {
    return getBaseSpeed() - getLoadFactor() * numberOfCoconuts;  }
}
class NorwegianBlue extends Bird {
  double getSpeed() {
    return (isNailed) ? 0 : getBaseSpeed(voltage);  }
}
// Somewhere in client code
speed = bird.getSpeed();
```



- پس از بازآرایی:

معرفی شیء پارامتر (INTRODUCE PARAMETER OBJECT)



- وقتی که تعدادی پارامتر معمولاً همراه هم پاس می‌شوند
- مثلا پارامترهای تاریخ شروع و تاریخ پایان در این مثال

سایر تکنیک های بازآرایی

- برای مطالعات بیشتر، به آدرس‌های زیر رجوع کنید:

1. http://refactoring.com/catalog/

- کاتالوگی از تکنیک های بازآرایی
- گردآوری شده توسط مارتین فاولر

2. https://refactoring.guru/refactoring/techniques

مطالب تکمیلی

استعاره دو کلاه

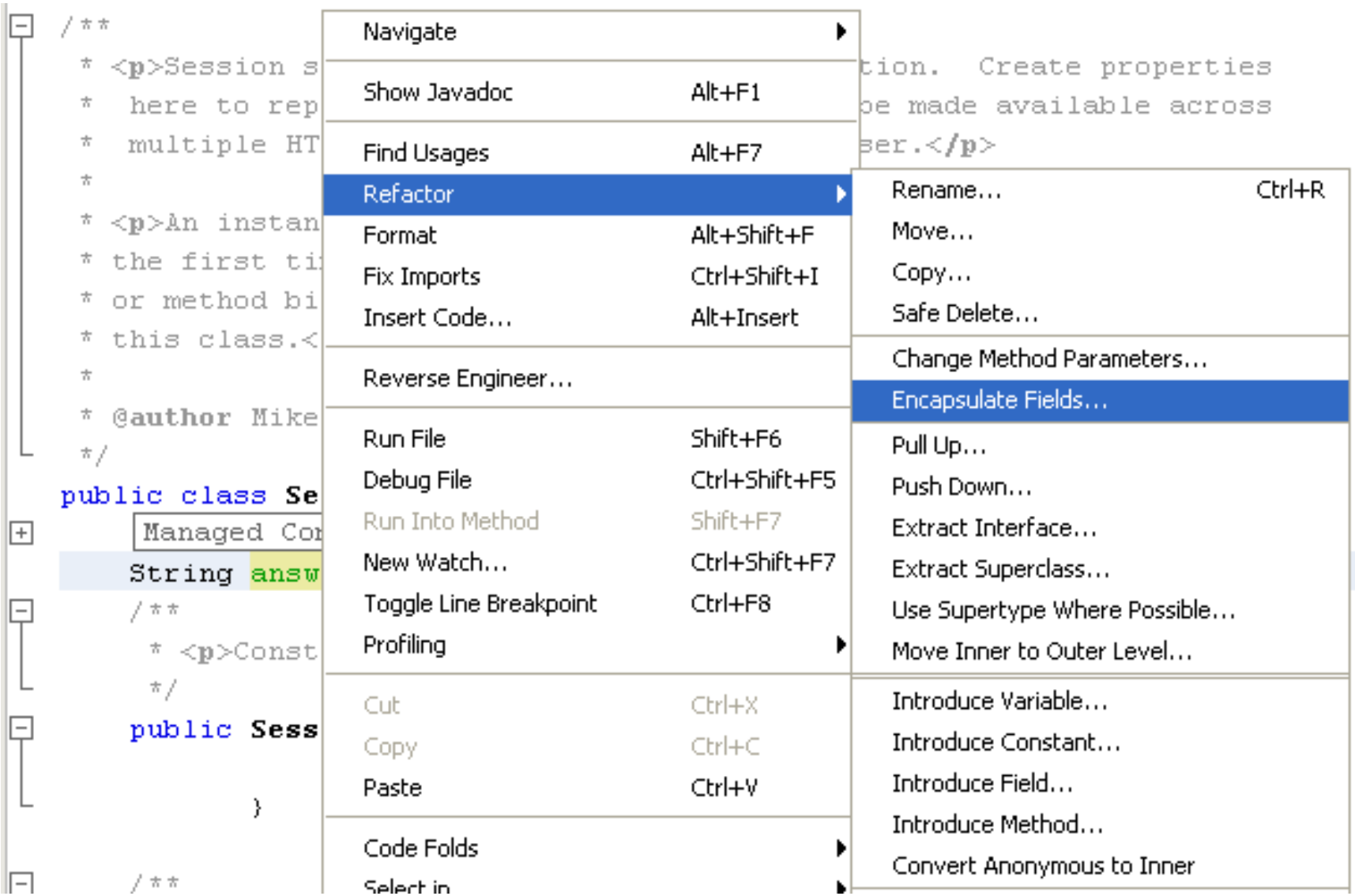
- در هنگام برنامه نویسی، زمان خود را به دو بخش **مجزا** تقسیم کنید:
  - تولید برنامه
  - بازآرایی

- 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 می کنند
- Eclipse, IntelliJ IDEA, NetBeans, ...
- اجرای تکنیک های بازآرایی را خودکار می‌کنند
- اشتباهات انسانی را کاهش می‌دهند
- و اجرای تکنیک ها 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

مثال: پشتیبانی Eclipse از بازآرایی



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

ریسک بازآرایی

- بازآرایی، ذاتاً مخاطره آمیز (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 لحاظ می‌شود**
- زمانی که صرف بازآرایی شده، تولید آینده شما را تسریع می‌کند

مهار خطر بازآرایی

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

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

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

جایگاه بازآرایی در متدولوژی های چابک

- بخشی مهم از متدولوژی‌های چابک
  - مثل XP
- 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

مراجع

- **Refactoring**: Improving the Design of Existing Code
  - Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts
    - کتابی قدیمی، ولی همچنان زنده و پرخواننده
- برخی صفحات مفید:

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

پایان

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 (با استفاده از بازآرایی)،
  - سپس در صورت لزوم آن را برای رسیدن به کارایی بهتر بهبود بخشید
- Write tunable software first and then tune it for sufficient speed

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