Access Control

کنترل دستیابی

مقدمه

- کنترل دسترسی(یا مخفی کردن پیاده سازی-implementation hiding) در مورد این است که "همه چیز در ابتدا در دسترس نباشد".
- این یکی از انگیزه های اصلی جهت refactoring است که در آن کدهای درون برنامه به صورتی که خواناتر،قابل فهم و قابل نگهداری باشند، بازنویسی می شوند.
- این موضوع در رابطه با کتابخانه ها بسیار مهم است.
- جاوا چند مشخصه دستیابی را فراهم کرده تا به ایجاد کننده کتابخانه امکان مشخص کردن این که چه چیزهایی در دسترس برنامه نویس client باشد و چه چیزهای در دسترس نباشد را بدهد.

واحد کتابخانهPackage

- یک package گروهی از کلاس هاست که همه باهم تحت یک namespace سازماندهی شده اند.

```
import java.util.ArrayList;
//or import java.util.*;
public class SingleImport {
  public static void main(String[] args) {
     ArrayList list = new java.util.ArrayList();
  }
}
```

سازماندهی کد

- به فایل source codeای را که برای جاوا ایجاد می کنیم، یک واحد کامپایل (گاهی اوقات واحد ترجمه)می گویند.
- هر کدام از واحدهای کامپایل باید دارای نامی باشند که به پسوند java.ختم شوند.در داخل واحد کامپایل می توان یک کلاس public داشت که باید همان نام را به عنوان یک فایل داشته باشد.
- پس از کامپایل هر کدام از فایل‌های جاوا خروجی با نام کلاس موجود با پسوند .class دارد.

سازماندهی کد

- یک برنامه کاری شامل گروهی از فایل های .class است که می توانند به صورت آرشیو شده و فشرده در یک فایل بایگانی جاوا(Java Archive= JAR) قرار گیرد(این کار از طریق jar archiver موجود در جاوا صورت می گیرد).
- مفسر جاوا در برابر پیدا کردن،لود کردن و تفسیر این فایل ها مسئول است.

برخورد (Collision)

- با توجه به اینکه نام دامنه اینترنتی به صورت تضمین شده یکتاست،در صورتی که از این قاعده پیروی کنیم،نام packageها نیز یکتا خواهد بود و هرگز تصادم نامی نخواهیم داشت.
- در صورتی که دو کتابخانه از طریق ‘*’ ،import شوند و شامل نامی یکسان باشند،چه اتفاقی می افتد؟به عنوان مثال کد زیر را در نظر بگیرید:

```
import com.demisco.simple.*;
import java.util.*;
```

هر دوی این پکیج ها کلاسی به نام Vector دارند

برخورد (Collision)

- در صورتی که بخواهید vectorای را ایجاد کنید برخورد اتفاق می افتد

```
Vector v = new Vector();
```

- برای مثال در مورد vector جاوا ،کامپایر نوشتن به این شیوه را ایراد می گیرد و ما را مجبور می کند تا به صورت صریح و روشن منظور خود را بیان کنیم:

```
java.util.Vector v = new java.util.Vector();
```

مشخصه های دسترسی جاوا-دسترسی package

- دسترسی پیش فرض هیچ کلیدواژه ای نداردو معمولا به دسترسی در سطح package اشاره دارد(به این سطح دسترسی friendly نیز میگویند).

این به این معنا است که همه کلاس های موجود در یک package منحصر به فرد قابلیت دسترسی به عنصر مد نظر را دارند اما کلاس های خارج از package این اجازه را ندارند و عنصر مد نظر نسبت به آن‌ها private است.

مشخصه های دسترسی جاوا-دسترسی public

- استفاده از کلیدواژه public به این معناست که عضوی که به صورت public تعریف شده توسط همه به ویژه برنامه نویس client که از کتابخانه مد نظر استفاده می کند،در دسترس است.

مشخصه های دسترسی جاوا-دسترسی private

- کلیدواژه private به این معناست که هیچ کس اجازه دسترسی به عضو مدنظر را (به جز کلاسی که عضو در آن قرار دارد) ندارد.

مشخصه های دسترسی جاوا-دسترسی protected:دسترسی ارث بری

- کلیدواژه protected با مفهومی به نام ارث بری در ارتباط است که از کلاس موجود که به عنوان کلاس پدر به آن نگاه می کنیم،گرفته شده است.
- گاهی اوقات سازنده کلاس پدر می خواهد اجازه دسترسی اعضای خاصی را به کلاس های مشتق شده از خودش و نه کل کلاس‌ها را بدهد. به بیانی دیگر اعضای نوع Protected می توانند تنها توسط اعضای داخل کلاس مورد نظر و یا انواع و اعضای مشتق شده از آن کلاس مورد دستیابی قرار بگیرند.

Access Specifiers	همان کلاس	همان پکیج	کلاس‌های ارث بری شده	بقیه ی پکیج ها
public	Y	Y	Y	Y
protected	Y	Y	Y	N
Default (Package Access)	Y	Y	N	N
private	Y	N	N	N

چند نکته

- تنها می توان یک کلاس public در واحد کامپایل(فایل) داشت.
- نام کلاس public باید دقیقا با نام فایل حاوی واحد کامپایل مطابقت داشته باشد.
- میتوان واحد کامپایلی داشت که هیچ کلاس publicای نداشته باشد ولی معمول نیست.
- یک کلاس نمی تواند به صورت private تعریف شود.

الگوی طراحی Singleton

- در صورتی که بخواهید کلاس دیگری به کلاس مد نظر دسترسی نداشته باشد،باید تمام constructorها را به صورت private تعریف کنید و از این طریق هیچ کس به جز شما نمی تواند داخل عضو static کلاس شی ای از کلاس را ایجاد کند.

```
class Soup {
    private Soup() {}
    private static Soup ps1 = new Soup();
    public static Soup makeSoup() {
     return ps1;
  }
}
```

متدMutator

- یک متد mutator متدی است که برای کنترل تغییرات یک متغیر استفاده می شود و این متدها متدهای setter نیز نامیده می شوند.معمولا هر setter با یک getter (که به آن accessor نیز می گویند)همراه است.
- با توجه به این اصل، متغیرهای عضو یک کلاس به صورت private تعریف شده اند تا از دسترسی کدهای دیگر مخفی بوده و از این طریق محافظت شوندو تنها توسط متدهای عضو public(متدهای mutator) قابل تغییر هستند.

Mutatorمتد

```
public class Tree {
    private int height;
    public int getHeight() {
        return name;
    }
    public void setHeight(int height) {
    this.height = height;
    }
}
```

متد Mutator

- دلایل استفاده از getter و setter
  - ✓ از این طریق می توان پیاده سازی را بدون تغییر واسط (interfaceها) تغییر داد.می توان قواعدی را برای setterها و getterهایمان تعریف کنیم.
  - ✓ همچنین در مواردی که بخواهیم با فیلد مد نظر به عنوان فقط خواندنی(تنها داشتن getter) و یا فقط نوشتنی(تنها داشتن setter)،رفتار کنیم،از این طریق امکان پذیر است.
  - ✓ بعضی از مجموعه ابزارهای مثل reflection و JavaBeans تنها objectهایی را می پذیرند که setter و getter داشته باشند.