پروژه های تمیز در جاوا

تا اینجا با مفاهیم اصلی برنامه نویسی شیءگرا در زبان جاوا آشنا شده ایم و کم و بیش با آن ها سر و کله زده ایم. در این قسمت قصد داریم تا با مرور این مفاهیم و البته معرفی برخی نکات جدید ارتباطات میان آن ها را کشف کنیم و ببینیم که چطور می توان با قرار دادن مناسبت اجزا در کنار یک دیگر پروژه های تمیزتری را ایجاد کنیم.

مروری برقواعد نام گذاری

پکیج (Package)

پکیج ها به ما این امکان را می دهند تا فایل هایی (کلاس ها، اینترفیس ها و ...) که به یکدیگر شبیه هستند یا با یکدیگر ارتباط دارند را در یک محل نگهداری و سازمان دهی کنیم.
در نام گذاری پکیج ها *تماما از حروف کوچک* استفاده می کنیم . اگر نام پکیج از چند واژه تشکیل شده باشد نیازی نیست از کاراکتر خاصی مثل - یا _ برای جدا سازی آن ها استفاده کنیم. به مثال های زیر تو توجه کنید:

```
com.example.deepspace    //Correct

com.exmaple.deep_space   //Improper
com.example.deepSpace    //Improper
```

کلاس (Class)

برای نام گذاری کلاس ها از سبک UpperCamelCase استفاده می کنیم. یعنی کلمات بدون هیچ فاصله یا جداکننده ای به صورت پشت سر هم ظاهر می شوند و هر واژه با حرف بزرگ آغاز شده و در ادامه آن از حروف کوچک استفاده می شود.
نام کلاس معمولا یک *اسم* یا یک *گروه اسمی* است. مانند `Character` یا `ImmutableList`.
نام اینترفیس (Inrerface) نیز معمولا شامل یک اسم یا گروه اسمی است اما گاهی نیز می تواند یک *صفت* یا *گروه وصفی* باشد مثل `Readable`.

متد (Method)

نام متدها به صورت lowerCamelCase نوشته می شوند. به عبارت دیگر کلمات بدون هیچ فاصله یا جداکننده ای به صورت پشت سر هم ظاهر می شوند و هر واژه **به جز اولین کلمه** با حرف بزرگ آغاز شده و در ادامه آن از حروف کوچک استفاده می شود.
نام متد معمولا یک *فعل* یا *گروه فعلی* است مانند `sendMessage` یا `stop`.

ثابت (Constant)

ثابت ها (Constants) متغیرهایی هستند که به صورت *static* و *final* تعریف شده اند. این متغیر ها با هدف *تغییرناپذیری (immutability)* تعریف می شوند، به عبارت دیگر تنها یک بار مقداردهی شده و نمی توان مقدار آن ها را تغییر داد. ثابت ها شامل موارد زیر می باشند:

- primitive types
- Strings
- immutable types
- immutable containers of immutable types

توجه داشته باشید که ثابت ها باید هنگام تعریف مقداردهی اولیه شوند.
نام ثابت به شکل CONSTANT_CASE نوشته می شود: همه حروف بزرگ بوده و هر کلمه از دیگری با کاراکتر_ جدا می شود.
مثال هایی را در ادامه مشاهده می کنید.

```
// Constants
static final int NUMBER = 5;
static final SomeMutableType[] EMPTY_ARRAY = {};
enum SomeEnum { ENUM_CONSTANT }

// Not constants
static String nonFinal = "non-final";
final String nonStatic = "non-static";
static final Set<String> mutableCollection =
new HashSet<String>(); //HashSet is a mutable collection

static final Logger logger = Logger.getLogger(MyClass.getName());
static final String[] nonEmptyArray = {"these", "can", "change"};
```

ساختار پروژه

به طور کلی ساختار یکتا و منحصر به فردی برای پروژه های جاوا وجود ندارد. با این حال به یکی از این ساختارهای پرکاربرد که در ابزار مدیریت پروژه Maven استفاده می شود اشاره می کنیم (اگر با این ابزار آشنایی ندارید به اینجا سر بزنید.)

- برای فایل های سورس(source files): src/main/java
  - برای فایل های منبع (text، image، xml و ...): src/main/resources
  - برای فایل های تست(test files): src/test/java
  - برای فایل های منبع تست (text، image، xml و ...): src/test/resources

ساختار فایل سورس (Source file structure)

هر فایل سورس از قسمت های زیر تشکیل شده است:

- گزاره پکیج (Package statement)
- گزاره های import
- دقیقا یک کلاس اصلی (که هم نام با اسم فایل است)

برای جداسازی هر یک از این اجزا از دیگری از یک *خط خالی* استفاده کنید.

گزاره های import

گزاره های import را به صورت زیر منظم کنید:

- import `java.*`
- import `javafx.*`
- import `others`

بین جدا کردن این دسته ها از یک *خط خالی* استفاده کنید.

سازمان مندی اجزای کلاس

مهم ترین نکته ای که در چیدمان اجزای یک کلاس باید به آن توجه کرد *داشتن یک سیر و ارتباط منطقی و طبیعی* است؛ به طوری که بتوانید به راحتی نحوه عملکرد یک فایل را برای کسی توضیح دهید.
مثلا سعی نکنید توابعی که دیرتر به کدتان اضافه می کنید را لزوما در انتهای فایل قرار دهید، بلکه بسته به نوع و عملکرد آن را در مناسب ترین محل تعریف و پیاده سازی کنید. برای یکدست و منظم کردن فایل سورس می توانید از الگوی زیر استفاده کنید:

```
public class CleanCode{
    // static fields
    // instance fields
    // constructor(s)
    // static methods
    // Getters and setters
    // method implementations coming from interfaces
    // instance methods
    // equals(), toString(), ...
}
```

و چند نکته دیگر!

جداسازی اجزا

استفاده از خطوط خالی برای جداسازی اجزای مختلف برنامه راهکار هوشمندانه ای برای نشان دادن ارتباط آنها با یک دیگر است.
در تعریف توابع سعی کنید به کمک خطوط خالی ارتباط توابع را مشخص کنید. اگر گروهی از توابع هستند که در *عملیات مشترکی* مورد استفاده قرار می گیرند(مثلا تمام توابعی که مربوط به پیاده سازی بخش شبکه هستند) بهتر است آنها را پشت سر هم تعریف کرده و بینشان 2 خط خالی قرار دهید. حال برای مجزا کردن این توابع از گروهی دیگر کافیست بین آنها از تعداد خطوط خالی بیشتری استفاده کنید.
همچنین توابعی که به یکدیگر وابسته بوده و یکدیگر را در مراحل پیاده سازی خود صدا می کنند را نیز بهتر است مشابه توضیحات بالا از دیگر
قسمت ها تفکیک کنید.

هر چه نزدیک تر بهتر

سعی کنید متغیرهای **محلی** را درست قبل از اولین استفاده آنها تعریف کنید. با تعریف یک باره تمامی این نوع متغیرها در ابتدای یک بلاک ممکن است در ادامه نوع آنها یا علت تعریفشان را فراموش کنید و نیاز باشد تا مکررا به ابتدای بلاک بازگردید.
همچنین اگر یک تابع، تابعی دیگر را صدا میزند، تابع **صداکننده باید درست بالای تابع صداشونده** قرار گیرد. رعایت این نکته کمک می کند تا کد یک رفتار و جریان طبیعی را دنبال کند.