به نام خدا

دستور کار کارگاه برنامه‌نویسی پیشرفته

نیم‌سال اول 97-98

جلسه سوم

تعامل بین اشیا و کار با package

مقدمه

جلسه قبل نحوه ساختن و کار با یک شی را یاد گرفتیم. اما برای ساختن یک برنامه کامل، ایجاد شی‌های که به طور مجزا کار می‌کنند، کافی نیست و باید ترکیبی از شی‌هایی را به وجود بیاوریم که با یکدیگر در تعامل هستند. برنامه‌ای که در این جلسه خواهیم نوشت، یک ساعت آنالوگ است که زمان را به صورت یک ساعت عقربه‌دار نمایش می‌دهد. علاوه بر کار با اشیا، در این جلسه قصد داریم تا کمی با نحوه استفاده دوباره از کد (code reuse) آشنا شویم. از این رو، قسمت‌هایی از کد این جلسه توسط تدریسیاران این درس به شما داده شده.

Modularization and Abstraction

در دنیای طراحی سیستم‌ها، برای حل مسائل، باید سعی کنیم طراحی مناسبی برای کلاس‌ها ارائه کنیم تا حل مسئله اصلی که پیچیدگی‌های زیادی دارد، به مدل‌سازی کلاس‌ها و حل مسائل کوچک‌تری تبدیل شود. شکستن مشکل اصلی به کلاس‌ها و متدهای کوچکتر علاوه بر ساده‌سازی مساله در فاز توسعه محصول و نگهداری آن نقش مهمی ایفا می‌کند. پیاده‌سازی برنامه در قالب کلاس‌ها و متدهای بزرگ باعث ایجاد مشکلاتی در مراحل بعدی توسعه و نگهداری نرم‌افزار خواهد شد.

راه حلی که معمولا برای مقابله با پیچیدگی وجود دارد، استفاده از abstraction و modularization است. ما همیشه سعی می‌کنیم مسائل را به مسئله‌های کوچک‌تر و هر مسئله کوچک‌شده را به چند مسئله ساده‌تر تبدیل کنیم تا جایی که حل هر مسئله مجزا، ساده و قابل انجام باشد. به این عمل modularization می‌گویند. در مرحله طراحی ماژول‌ها، مساله به چند زیرمساله می‌شکنند که تا جای ممکن از یکدیگر مستقل هستند. این مستقل بودن باعث می‌شود افراد تیم توسعه بصورت جداگانه روی مساله خود تمرکز کنند و نیازی به درنظر گرفتن قسمت‌های دیگر پروژه نباشند. برای ارتباط بین ماژول‌ها با یکدیگر از واسط‌هایی استفاده می‌شود که به آن‌ها واسط‌های[1] آن module گفته می‌شود. همچنین وقتی مساله را به مسایل مستقل از یکدیگر تقسیم می‌کنیم، نیازی به دانستن نحوه‌ی

[1] Interface

پیاده‌سازی آن قسمت از کد را نداریم و فقط واسط‌های آن module (می‌تواند کلاس، متد، پکیج و... باشد) اهمیت پیدا می‌کند. به عمل در نظر گرفتن کلاس‌ها به عنوان یک واحد و عدم توجه به جزئیاتی که کاربر کلاس به آنها نیاز ندارد abstraction می‌گویند.

به عنوان مثال یک سامانه رزرو بلیط سینما را درنظر بگیرید. این سامانه از چندین قسمت مهم تشکیل شده که عبارتند از:

- مدیریت کاربران سامانه
- مدیریت سانس‌های سینما و صندلی‌های خالی
- فروش بلیط به کاربران

این سه قسمت بصورت کلی سامانه رزرو بلیط را تشکیل می‌دهند. در مرحله طراحی در ابتدا وظیفه هر قسمت مشخص می‌شود. به عنوان مثال قسمت مدیریت کاربران سامانه وظیفه احراز هویت، ورود و خروج کاربران را دارد. قسمت فروش بلیط نیز باید مبلغ درخواستی را از کاربر بگیرد، صندلی‌های درخواستی کاربر را از نظر خالی بودن بررسی کند، و در صورت خالی بودن صندلی ها مبلغ بلیط(های) سفارش داده شده را از حساب کاربر کسر کند و پرینت بلیط‌ها را به او بدهد. حال کسی که بر روی قسمت فروش بلیط کار می‌کند نیازی به پیاده‌سازی نحوه تصدیق کاربر ندارد. از دید این توسعه‌گر نحوه‌ی ارتباط با قسمت مدیریت کاربر اهمیت دارد. به عنوان مثال مدیریت کاربران باید متدهایی داشته باشند که بگوید آیا کاربر در سایت لاگین هست یا خیر و یا اینکه کاربری که درخواست خرید را به سایت داده چه کسی است. از این رو توسعه‌گر بخش مدیریت کاربر، پیاده‌سازی قسمت خود را از دید توسعه‌گرهای قسمت‌های دیگر پنهان می‌کند و یک سری واسط‌هایی طراحی می‌کند که بقیه بتوانند از سرویس او استفاده کنند.

Package

در کلاس درس شما با مفهوم کلاس آشنا شدید. تقسیم پروژه به کلاس‌ها یکی از روش‌های abstraction است. در اینجا شما با private کردن یک سری متد، سایر افرادی که از کلاس شما استفاده می‌کنند را از پیاده‌سازی داخلی کلاس خود بی‌خبر می‌کنید. همچینی با طراحی متدهای public قابلیت ارتباط کلاس خود با دیگر قسمت‌های پروژه را فراهم می‌کنید. یکی دیگر از راه‌های abstraction استفاده از package است. package این قابلیت را به توسعه‌گرها می‌دهد تا تعدادی از کلاس‌های خود را در اختیار قسمت‌های دیگر قرار ندهند. به عنوان مثال طراح قسمت مدیریت کاربر تعدادی کلاس برای ارتباط با پایگاه داده می‌نویسد تا اطلاعات کاربران (نام کاربری، گذرواژه و ...) را بازیابی و ذخیره کند. این کلاس داخل قسمت مدیریت کاربر تعریف شده و خارج از این قسمت نیازی به دسترسی مستقیم به آنها نیست (بقیه قسمت‌های کد برای گرفتن اطلاعات کاربران باید آنرا از قسمت مدیریت کاربران درخواست

کنند). همچنین در توسعه یک محصول نرم‌افزاری تعداد کلاس‌ها بسیار زیاد می‌شود. برای سادگی کار کلاس‌های مربوط به یکدیگر را در کنار هم قرار داد تا توسعه محصول ساده‌تر گردد (همانند فولدر بندی فایل‌ها کامپیوتر). به روایتی پکیج نیز می‌تواند برای فایل‌های کلاس نقش فولدر را بازی کند.

انجام دهید

ابتدا پروژه درس را بسازید و همانند دستورکار قبلی آنرا با git مدیریت کنید. همانطور که در ابتدا گفته شد در این قسمت می‌خواهیم یک ساعت آنالوگ را پیاده‌سازی کنیم. این برنامه از سه قسمت تشکیل شده:

- قسمت مربوط به منطق ساعت
- قسمت مربوط به نمایش ساعت عقربه‌ای
- قسمت مرتبط دهنده دو قسمت بالا

پیاده‌سازی قسمت دوم آن در قالب پکیج org.clock.ui در اختیار شما قرار گرفته. پکیج را از moodle دانلود کرده و بعد از extract کردن، پوشه‌ی org را به همراه کلاس‌های آن، داخل پوشه src کپی کنید. بعد از افزودن پکیج باید آنرا آزمایش کنید. معمولا پکیج‌ها یک سری مستنداتی دارند که نحوه‌ی کار با آن پکیج را توضیح می‌دهد. در این جلسه، مستندات در دستورکار قابل مشاهده است.

برای نمایش ساعت باید یک شی از کلاس ClockUI ساخته شود. این کلاس دو constructor دارد.

```
public ClockUI();
```

این سازنده یک شی از کلاس ClockUI را می‌سازد و زمان آنرا 00:00:00 می‌گذارد.

```
public ClockUI(int hour, int minute, int second);
```

این سازنده یک شی از کلاس ClockUI را می‌سازد و زمان آنرا hour:minute:second می‌گذارد.

```
public void setClock(int hour, int minute, int second);
```

این تابع زمان ساعت را به hour:minute:second تغییر می‌دهد.

```
public void setHour(int hour);
```

مقدار ساعت را در زمان تغییر می‌دهد.

```
public void setMinute(int minute);
```

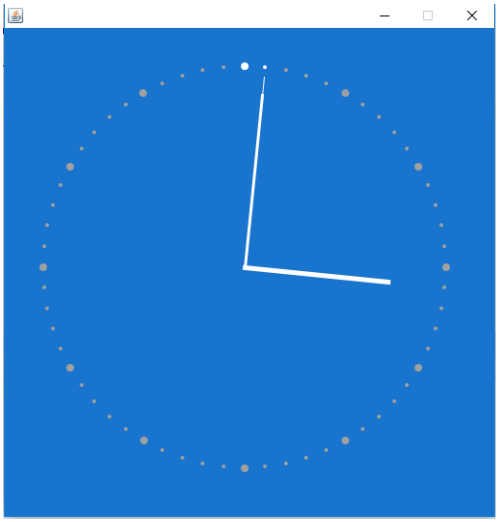
مقدار دقیقه را در زمان تغییر می‌دهد.

```
public void setSecond(int second);
```

مقدار ثانیه را در زمان تغییر می‌دهد.

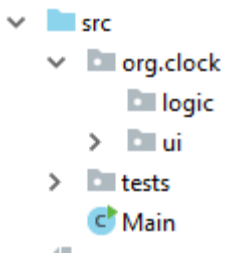
بعد از افزودن پکیج تغییرات خود را commit کنید. از این قسمت به بعد پیام‌های commit را خودتان تعیین کنید و پیام مناسبی برای تغییراتی که دادید بنویسید.

حال می‌خواهیم این پکیج جدید را آزمایش کنیم. برای این کار یک پکیج جداگانه به نام test درست خواهیم کرد که این پکیج حاوی فایل‌های تست برنامه است. جدا کردن این پکیج از سایر پکیج‌ها باعث شده تا کدهای مربوط به بخش تست از کدهای توسعه محصول جدا باشد. خارج از پکیج org در پوشه src یک پکیج دیگری به نام test درست کنید. برای این کار روی پوشه src راست کلیک کنید. به گزینه‌ی new رفته و package را انتخاب کنید و نام آنرا بنویسید. سپس یک کلاس برای تست ساعت به عنوان ClockUITest بسازید. در داخل متد main آن یک شی از کلاس ClockUI ساخته و آنرا با استفاده از توابع setClock, setHour, setMinute, setSecond مقداردهی کنید. از آنجایی که در اینجا خروجی، یک پنجره نمایش دادنی است، خروجی کلاس تست شما نمی‌تواند true یا false باشد و شما فقط می‌توانید اجراشدن کد را بسنجید. تغییرات خود را commit کنید.



شکل 1 نمایش ساعت آنالوگ

حال می‌خواهیم قسمت مربوط به منطق را پیاده کنیم. برای این کار باز یک پکیج دیگر ساخته و نام آنرا org.clock.logic بگذارید. خواهید دید که این پکیج در کنار پکیج UI و در زیر پکیج org.clock نمایش داده می‌شود. حال کلاس ClockLogic را داخل پکیج org.clock.logic بسازید و توابع زیر را به آن اضافه کنید.



شکل 2 نمایش ساختار package های پروژه

```
public ClockLogic(int hour, int minute, int second);
```

این تابع سازنده زمان ساعت را به hour:minute:second تغییر می‌دهد.

```
public void setClock(int hour, int minute, int second);
```

این تابع زمان ساعت را در صورتی که بعد از ساخته شدن شی نیاز به تغییر باشد، به hour:minute:second تغییر می‌دهد.

```
public void tik(int second);
```

این تابع زمان را به اندازه‌ی مقدار ورودی جلو می‌برد. واحد ورودی ثانیه است.

```
public int getHour();
```

مقدار ساعت را بازمی‌گرداند.

```
public int getMinute();
```

مقدار دقیقه را بازمی‌گرداند.

```
public int getSecond();
```

مقدار ثانیه را بازمی‌گرداند.

دقت کنید که فرمت خروجی توابع getHour و getMinute و getSecond باید اعداد صحیح بین 0 تا 60 باشند. تبدیل زمان تغییریافته بوسیله تابع tik و افزودن دقیقه و ساعت بر اساس آن برعهده این کلاس است. مثلا اگر زمان هست 01:01:00، با صدا زدن تابع tik(100)، زمان باید به 01:02:40 تغییر کند. فرمت ساعت را 24 ساعته درنظر بگیرید.

همانند جلسه‌ی قبل می‌خواهیم درستی متدهای کلاس ClockLogic را بررسی کنیم. فایل ClockLogicTest.java درستی کلاس ClockLogic بررسی می‌کند. آنرا دانلود کنید و به پکیج test خود اضافه کنید. سپس به پیاده‌سازی توابع بپردازید تا تمامی خروجی‌های تابع main این کلاس برابر true شود. سپس تغییرات خود را commit کنید.بعد از آنکه از درستی کلاس‌های ClockLogic و ClockUI اطمینان پیدا کردید، در یک کلاس جداگانه ایندو را به یکدیگر مرتبط کنید. در روش های طراحی، فایلی که نقطه‌ی آغاز پروژه است معمولا در پکیجی قرار نمیگیرد و در پوشه اصلی پروژه ایجاد می‌شود. در پوشه src یک کلاس Clock ایجاد کنید. این کلاس باید از دو کلاس ClockLogic و

ClockUI استفاده کند و بین آنها ارتباط برقرار کند. برای ایجاد ارتباط ابتدا داخل متد main کلاس Clock از دو کلاس بالا یک شی بسازید. سپس در این متد یک for بنویسدی که تابع ()tik شی ساخته شده از کلاس ClockLogic را فراخوانی کند و زمان خروجی کلاس ClockLogic را به متد setClock شی ساخته شده از کلاس ClockUI بدهید. با جستجو در اینترنت برنامه را طوری بنویسید که کدهای داخل حلقه با یک ثانیه تاخیر اجرا شود تا خروجی شبیه یک ساعت آنالوگ واقعی شود. در نهایت تغییرات خود را commit کنید.

در نسخه بعدی این ساعت قصد داریم تا زمان چندین شهر را نمایش دهیم. برای پیاده‌سازی این کار ابتدا برای هر شهر یک اختلاف زمانی[2] تعریف می‌کنیم که برابر است با فاصله زمانی از ساعت گرینویچ. این اختلاف زمانی می‌تواند مثبت یا منفی باشد. سپس باید زمان گرینویچ را با این عدد جمع یا تفریق کرد.

در این قسمت داخل پکیج org.clock.logic یک کلاس TimeZone بنویسید که سازنده آن مطابق قطعه کد زیر است.

```
public TimeZone(int hour, int minute, int positive);
```

در اینجا hour و minute اختلاف ساعت و دقیق از ساعت گرینویچ هستند. positive نیز بیانگر آن است که آیا این اختلاف زمانی باید با ساعت گرینویچ جمع شود یا تفریق شود. در حالت اول مقدار 1 و در حالت دوم مقدار 1- می‌پذیرد. به عنوان مثال برای تهران داریم:

```
TimeZone tehranTimeZone = new TimeZone(3, 30, 1);
```

حال باید ClockLogic را بگونه‌ای تغییر دهید که قابلیت اعمال اختلاف زمانی را داشته باشد. برای این کار یک سازنده دیگری بسازید با مشخصات زیر:

```
public ClockLogic(int hour, int minute, int second, TimeZone tz);
```

سپس داخل این تابع اختلاف زمانی رو روی hour و minute اعمال کنید (آنها را با مقدار hour و minute شی tz جمع یا تفریق کنید.

نکته: دقت کنید که این کار را نیز در تابع setClock نیز باید انجام دهید چرا که در آنجا نیز باید زمان را برروز کنید.

نکته: دقت کنید ممکن است با تفریق کردن اعداد ساعت منفی شود. در صورتی که ساعت منفی شد باید آنرا با 24 جمع کرد. در صورتی که دقیقه منفی شد، یک ساعت کم کنید و دقیقه را با 60 جمع کنید.

[2] Time-zone

در تابع main کلاس Clock علاوه بر ساعت قبلی که وقت گرینویچ را نمایش می‌داد دو ساعت دیگر اضافه کنید که به زمان‌های تهران و نیویورک را نمایش می‌دهد. اختلاف زمانی تهران و نیویورک به ترتیب 3:30+ و 5:00- است. سپس هر کدام را نمایش دهید (هر شی از کلاس ClockUI یک ساعت را نمایش می‌دهد).