امیرحسین ایروانی‌منش 9931066 airavanimanesh@aut.ac.ir

تکلیف شماره ۲

تمرین شماره ۱

الف) در هر شئ State به معنای ویژگی و مشخصاتی که دارد اشاره میکند، برای مثلا برای یک شئ از یخچال، اندازه، مصرف انرژی، نوع کابری و ... دارد. Behavior در هر شئ به عملگر ها و امکانات هر شئ اشاره میکند برای مثال یک یخچال میتواند دمای محتویات درون خود را پایین بیارد یا مثلا دمای داخلش را روی مانیتورش نشان بدهد و در آخر Identity در هر شئ به هویت آن شئ اشاره میکند، برای مثال یک یخچال نمی تواند یک تلویزیون یا یک اتومبیل باشد.

ب) به طور معمول در برنامه هایی که با جاوا نوشته میشوند، اشیا ی مختلفی از الگو ها و تمپلیت هایی که با نام کلاس آنها را می‌شناسیم ساخته می شوند. این اشیا با ارسال پیام به یکدیگر در تعامل هستند، که نتیجه هر پیام فراخوانی یک متدی است که عملیاتی رو انجام میدهد یا وضعیت شئ دریافتی به عنوان پارامتر را تغییر می‌دهد.

ج) Overloading به ما این اجازه را می دهد که چند متد همنام بسازیم که مشخصه های متفاوتی باشند، این مشخصه میتواند تعداد آرگومان های ورودی و یا نوع ارگومان های ورودی و یا هر دوی آن ها باشند، در overloading میتوانیم یک متد با مشخصه متفاوت و همچنین return type متفاوت ایجاد کنیم.

casting در جاوا به معنای این است که نوع یک متغیر تغییر کند که به صورت کلی دو نوع است، کستینگ واگرا (اتوماتیک) و کستینگ همگرا (دستی). برای مثال اگر در یک متغیر از نوع اینتجر عدد ۱۰ را بریزیم و ان را داخل متغیری از نوع دابل بریزیم، به طور خودکار اینتجر به دابل کست میشود (کستینگ واگرا). در کستینگ همگرا باید به صورت دستی در پرانتز نوع داده کوچکتری را که میخواهیم را بنویسیم تا کستینگ اتفاق بیوفتد، در زیر لیستی از انواع داده های اصلی (primitive) با ترتیب سایز امده است:

`byte -> short -> char -> int -> long -> float -> double`

modularization به طور کلی یک طرح برای برنامه نویسی است که در ان برنامه به ماژول های مختلف تقسیم شده که اینکار در حل مسئله کمک شایانی میکند. برنامه نویسی شئ گرا، دیدگاهی همراه و همراستا با modularization است. در این طرح یک مسئله به مسئله های کوچک تر تقسیم شده و جدا جدا حل و ازمون و خطا میشوند تا به نتیجه دلخواه برسند و در اخر مسئله اصلی حل می شود.

abstract در جاوا ویژگی ای است که باعث میشود فقط جزئیات ضروری و اساسی برای کاربر نمایش داده شوند و جزئیات پیش پا افتاده و غیر ضروری از چشم کاربر دور بماند. برای مثال به یک یخچال به عنوان یک "یخچال" نگاه میکنند نه به عنوان چیزی که با یک سری قطعات خاص عملیات خنک سازی را انجام می دهد. یک مثال دیگر اینکه وقتی یک شخص در ماشین گاز میدهد میداند که اگر گاز بدهد سرعت ماشین زیاد میشود ولی به طور دقیق نمی داند چه فرآیندی طی می شود تا سرعت ماشین زیاد شود.

د) Immutable object در جاوا به معنی یک شئ immutable از یک کلاس immutable هست که لفظ immutable به این معنی است که بعد از ساخته شدن شئ، امکان تغییر مقادیر فیلد های ان وجود ندارد. برای مثال وقتی در یک فروشگاه برای یک کالا بارکدی تعیین میشود دیگر نباید بتوان ان را تغییر داد پس یک فیلد immutable با کیوورد final میسازیم که دیگر نتوان مقدار آن را تغییر داد.

تمرین شماره ۲

۱. غلط. کامپایلر در صورتی که کانستراکتوری به صورت دستی نوشته شده باشد، کانستراکتور پیش فرضی ارائه نمیدهد.

۲. صحیح. ArrayList به دلیل اینکه یک سری فعل و انفعالات برای چک کردن مرز های آرایه و تداخل های تصادفی را انجام میدهد کمی کند تر از array معمولی است.

۳. غلط. همه ی کلاس های کالکشن در جاوا مقادیری از آبجکت ها نگهداری میکنند که در آن primitive type ها جای نمی گیرند، برای استفاده مشابه از آنها میتوان از کلاس های Integer, Boolean, Double, Byte و ... استفاده کرد.

۴. صحیح. برای هر کلاس میتوان کانستراکتور های متفاوت و متعددی با پارامتر های ورودی متفاوتی ایجاد کرد (به اصطلاح Constructor Overloading).

۵. صحیح. در متد و کانستراکتور های هر کلاس استفاده از کیوورد this به مقادیر متغیر های همان کلاس اشاره دارد.

۶. صحیح. اگر به یک متغیر لوکال مقداردهی اولیه داده نشود برنامه دچار ارور کامپایل خواهد شد.

۷. غلط. اگر در یک متد یک متغیر خاصی تعریف شود به آن local variable میگوییم و فقط در همان متد قابل دسترسی هستند.

۸. صحیح. اگر در کانستراکتور مقداردهی ای انجام شود، مقدار های اولیه کلاس تغییر پیدا می کنند.

تمرین شماره ۳

الف)

۱. وقتی تابعی را اورلود میکنیم پارامتر های ورودی آن باید با تابع parent متفاوت باشد، که در این صورت داشتن تابع اورلود شده (که پارامتر های آن متفاوت است) با return type متفاوت امکان پذیر است. ولی اگر پارامتر ها یکسان باشند همچین چیزی امکان ندارد و خطای کامپایل ایجاد میکند.

۲. از کانستراکتور برای مقدار دهی اولیه یک شئ استفاده می شود در حالی که از متد برای نمایش توابع و functionality یک شئ استفاده می شود. کانستراکتور ها نمی توانند چیزی را برگردانند (return type ندارند) ولی متد ها باید ریترن تایپ مشخصی داشته باشند. در صورت عدم وجود کانستراکتور، کامپایلر به صورت پیشفرض یک کانستراکتور استفاده میکند که همچین چیزی برای متد ها صادق نیست. کانستراکتور حتما باید همنام با کلاس باشد در صورتی که متد میتواند هر نام دلخواهی باشد. کانستراکتور به صورت ضمنی و خودکار توسط سیستم صدا زده می شود در صورتی که متد توسط برنامه نویس و کاربر صدا زده میشوند.

۳. بله، یک کلاس میتواند چند کانستراکتور با پارامتر های ورودی متفاوت باشند. برای اورلود کردن کانستراکتور باید یک کانستراکتور مانند کانستراکتور قبلی ساخت اما با پارامتر های ورودی متفاوت. در اورلود کردن کانستراکتور ها میتوان از کیوورد this() استفاده کرد، که با قرار دادن پارامتر ورودی در پرانتز، به کانستراکتوری که چنین ورودی هایی میگیرد برنامه را ارجاع میدهیم.

ب) اولین ایراد در خط اول تابع main هستش که به کانستراکتور کلاس person فرمت متناقض پاس داده شده، برای رفع این مشکل باید در کد کانستراکتور کلاس person در گرفتن id به جای int از String استفاده کنیم.

ایراد دیگر در همان بخش این است که برای کانستراکتور کلاس person، return type مشخص شده که باید برداشته شود. (به طور کلی کانستراکتور return type ندارد)

در خط اول کلاس person نام Firstname از قاعده camelCase پیروی نمی کند و در نتیجه باید به firstName تغییر پیدا کند.

در خط دوم کلاس person نیز نام last_name از قاعده camelCase پیروی نمی کند و باید به lastName تغییر پیدا کند.

به دلیل تغییری که در ورودی کانستراکتور ایجاد کردیم در خط سوم کلاس person نیز باید تایپ id را String کنیم.

سپس به دلیل درست کردن نام گذاری متغیر ها برای اینکه در کانستراکتور اختلالی ایجاد نکند باید قبل از هر ۳ متغیر (نام، نام خانوادگی، و آیدی) کیوورد this گذاشته شود مثل this.firstName

در تابع express که یک ورودی String میگیرد، return type تابع از نوع اینتجر تعریف شده که با توجه به نوع استفاده از آن، وجود آن ضروری نبوده و میتوان آن را به void تغییر داد.

در تابع print باید تغییراتی که در نامگذاری ایجاد کرده ایم رو دوباره انجام دهیم.

** به دلیل تغییر نکردن فیلد های نام، نام خانوادگی و id و immutable بودن آنها میتوان آنها را از نوع final تعریف کرد.

```
public class Main {
    public static void main(String[] args) {
        person p1 = new person("Ted", "Mosby", "123456");
        p1.express();
        p1.express("Happy");
        p1.print();
    }

    static class person {
        private final String firstName;
        private final String lastName;
        private final String id;

        public person(String firstName, String lastName, String id) {
            this.firstName = firstName;
            this.lastName = lastName;
            this.id = id;
        }

        public void express() {
            System.out.println("I feel neutral");
        }

        public void express(String state) {
            System.out.println("I feel " + state + " today");
        }

        public void print() {
            System.out.println("person{" +
                    "name='" + firstName + '\'' +
                    ", lname='" + lastName + '\'' +
                    ", id=" + id +
                    '}');
        }
    }
}
```

امیرحسین ایروانی‌منش 9931066 airavanimanesh@aut.ac.ir

تمرین شماره ۴

۱. در آرایه معمولی خانه هایی از حافظه کنار هم شامل مقادیر مختلفی میشوند که کل آرایه دارای سایز استاتیک و ثابت است و به همین دلیل اضافه کردن مقدار جدید به آرایه و یا حذف یک خانه از آرایه امکان پذیر نیست. ساختار داده لینکدلیست به این صورت است که هر خانه آن به اصطلاح یک نود نامگذاری میشوند و در هر نود آدرس خانه بعدی (در بعضی موارد خانه قبلی نیز هم) و مقدار وجود دارد. برای پیمایش در این نوع ساختار داده از نود اصلی و اولیه (head) تا خانه ی مورد نظر پیمایش میشود و این یکی از دلایلیست که کاربری و عملکرد لینکدلیست را تا حدی کند میکند.در این ساختار داده بر خلاف آرایه معمولی امکان اضافه یا حذف یک خانه امکان پذیر است. در ساختار داده ی ArrayList امکانی است که به ما یک array با حافظه داینامیک میدهد ولی باز هم به سرعت آرایه معمولی نیست. در این ساختار داده هر وقت خانه ای اضافه میکنیم که بیشتر از سایز فعلی arrayList باشد، به صورت خودکار آرایه ای جدید تشکیل میدهد و آرایه فعلی را کپی میکند. برای مقایسه: یک ساختار داده لینکدلیست از تعدادی نود و در arraylist از یک آرایه داینامیک و پویا استفاده میشود و در آرایه معمولی نیز از خانه های خود حافظه به صورت ساده استفاده میشود. Arraylist و linkedlist توانایی پشتیبانی از انواع آبجکت و شئ های مختلف را دارند. نفوذ و ایجاد تغییر در لینکدلیست سرعت بیشتری نسبت به arraylist دارد به دلیل اینکه در صورت حذف یا اضافه آیتم جدید یک خانه اضافه می شود و مانند arraylist نیاز به شیفت دادن مقادیر و ... نیست. به طور کلی لینکدلیست در نفوذ و دسترسی به داده ها عملکرد بهتری نسبت به arraylist دارد در صورتی که arraylist در چیدمان و دسترسی داده ها عملکرد بهتری دارد.

۲. استفاده از Arraylist بهینه تر است. array که اصلا قابلیت اضافه و حذف داده ندارند که به کنار. در لینکدلیست برای اضافه کردن یک خانه باید ۱ عملیات انجام شود (به سر لیست یا ته لیست اضافه شود) و برای دسترسی باید k بار (k شماره خانه مورد نظر است) و برای حذف کردن یک بیمار هم باید k بار عملیات انجام شود، در صورتی که برای arraylist برای اضافه و دسترسی تنها ۱ عملیات کافی است و فقط برای حذف یک بیمار نیاز به k عملیات داریم.

۳. بله در arraylist امکان اضافه کردن چند باره ی یک آبجکت وجود دارد و arraylist اقدامی برای شناسایی آبجکت های تکراری انجام نمی دهد.

۴. در arraylist متد size تعداد آبجکت های درون آن را میدهد در صورتی که متد length برای آرایه، طول و حجم مد نظر قرار گرفته شده آرایه را نشان میدهد که در initialize آرایه تعیین می شود و ربطی به اشغال شدن فضاهایش یا نشدنش ندارد.