به نام خدا

تمرین دوم درس برنامه نویسی پیشرفته

نام و نام خانوادگی: امیرحسام ادیبی نیا

شماره دانشجویی: ۹۹۳۱۰۸۷

ترم زمستان ۰۰ - ۹۹

سوال اول

- **الف)**
  - تمام شئ‌ها دارای سه مشخصه هستند که برای تعریف کردن آن شئ، لازم است که آن‌ها را مشخص کنیم:
    - State: استیت یک شئ نشانگر مشخصات و خواص آن شئ است. به عبارتی دیگر، استیت یک شئ، بیانگر این است که آن شئ چه ویژگی‌هایی دارد. برای مثال، یک ماشین دارای سرعت، رنگ، اندازه، مدل و ... است.
    - Behavior: رفتار یک شئ، نشانگر کارهایی است که آن شئ، انجام می‌دهد. برای مثال، یک ماشین می‌تواند حرکت کند، دنده عوض کند، روشن شود و ... .
    - Identity: هویت یک شئ، شامل تمام ویژگی‌هایی است که آن را از باقی شئ‌ها متمایز می‌کند. برای مثال، یک ماشین دارای پلاک، شماره شاسی و ... است که آن را از باقی ماشین‌ها تمیز می‌دهد.
- **ب)**
  - به طور کلی تعامل اشیاء را می‌توان به دو دسته‌ی اصلی تقسیم نمود. دسته‌ی اول شامل تمام تعاملات یک طرفه‌ای است که یک شئ، با یک یا چند شئ دیگر انجام می‌دهد. برای مثال ارسال فایل از کامپیوتر به یک یا چند پرینتر، مثالی از این دسته است. دسته‌ی دوم اما شامل تعاملاتی است که به صورت دو طرفه انجام می‌شود. برای مثال هنگامی که کامپیوتری یک درخواست برای یک وب‌سایت می‌فرستد و منتظر پاسخ‌گویی می‌ماند، این نوع تعامل رخ داده است. (منبع)
- **ج)**
  - Overloading: اورلود کردن یک متد، به این معنا است که دو یا چند متد دارای اسم یکسان، ورودی‌ها و خروجی متفاوت داشته باشند.

- Casting: کست کردن به این معنا است که نوع یک متغیر ابتدایی را تغییر دهیم. برای مثال هنگامی که یک متغیر از نوع double را برابر با متغیری از جنس int می‌گذاریم، برنامه ابتدا نوع متغیر دوم را به double کست می‌کند و سپس آن را در متغیر اول ذخیره می‌کند.
- Modularization: مودل‌سازی، فرآیندی است که در آن مسئله به تعدادی مودل (کلاس) تقسیم می‌شود و پس از آن، هر مودل پیاده‌سازی می‌شود و به این ترتیب، کل مسئله حل می‌شود.
- Abstraction: ابسترکشن، مفهومی است که تنها ویژگی‌های اصلی یک کلاس را قابل دسترس قرار می‌دهد و باقی اطلاعات را برای خارج کلاس، پنهان نگه می‌دارد.

- **د)**
  - اشیاء غیر قابل تغییر، اشیاء ای هستند که پس از ساخته شدن، دیگر قابل تغییر نیستند. برای مثال، یک شئ از کلاس String در زبان جاوا، غیر قابل تغییر است. زیرا پس از آنکه ساخته شود، دیگر قابل تغییر نیست.

سوال دوم

- ۱)
  - غلط است. جاوا تنها هنگامی که هیچ کانستراکتور ای برای یک کلاس تعریف نشده باشد، یک کانستراکتور پیش‌فرض درست می‌کند.
- ۲)
  - صحیح است. در ArrayList، سایز آرایه متغیر است و ممکن است در صورت اضافه کردن عضو در آرایه، هزینه‌ی زیادی صرف شود.
- ۳)
  - غلط است. در ArrayList تنها می‌توان اشیاء از یک جنس را نگه داشت و نباید شامل نوع‌های ابتدایی[1] باشد.
- ۴)
  - صحیح است. می‌توان چند کانستراکتور در یک کلاس تعریف کرد، تنها باید این شرط را رعایت کرد که ورودی کانستراکتورها متفاوت باشد و دقیقاً مانند هم نباشند. اصطلاحاً به این حالت Overload کردن می‌گویند.
- ۵)
  - صحیح است. با دستور this در متد یا کانستراکتور یک کلاس، می‌توان به شئ ای که متد روی آن صدا زده شده، دست یافت.
- ۶)
  - صحیح است. در جاوا متغیرهای محلی[2]، پیش از استفاده باید مقداردهی شوند، در غیر این صورت برنامه کامپایل نمی‌شود.

[1] Primitive Types
[2] Local Variables

- **۷)**
  - غلط است. متغیرهایی تعریف شده در یک کلاس و خارج از متدها، instance variables نامیده می‌شوند و می‌توان آن‌ها را در تمام متدها استفاده نمود.
- **۸)**
  - صحیح است. هنگامی که یک شئ از یک کلاس ساخته می‌شود، مقادیر فیلدهای آن مقدار پیش‌فرضی دارند که پس از فراخوانی کانستراکتور، عوض می‌شوند و مقدار جدید را نگه می‌دارند.

سوال سوم

- **الف)**
  - **۱.** خیر؛ در صورت overload کردن یک متد، حتماً باید ورودی‌های متدها با هم متفاوت باشند (یا در تعداد، یا در ترتیب و نوع). در صورتی که ورودی‌ها با هم فرق کند، نوع خروجی‌ها هم می‌تواند متفاوت باشند.
  - **۲.** کانستراکتورها به منظور مقداردهی اولیه هنگام ساخت اشیاء یک کلاس استفاده می‌شوند. در صورتی که متدها به منظور اضافه کردن یک کاربرد به کلاس هستند و وظایف یک شئ را پیاده‌سازی می‌کنند.
  - **۳.** بله؛ در جاوا می‌توان کانستراکتورها را overload کرد. فقط باید این شرط را رعایت کرد که توابع استفاده شده به عنوان کانستراکتور، باید در ورودی با هم متفاوت باشند (یا در تعداد، یا در ترتیب و نوع).
- **ب)**
  - در خط سوم، هنگام ساختن یک instance از کلاس person، باید به نوع‌های ورودی کانستراکتور هنگام ساختن یک نمونه از آن توجه کرد. در کانستراکتور ساخته شده، ورودی سوم از نوع int است، در صورتی که در خط سوم، String ورودی داده شده است. در خط ۱۲ نیز، از آنجا که کانتراکتور هیچ خروجی‌ای ندارد، نباید نوع خروجی را معلوم کنیم. همچنین در خود کانستراکتور، ورودی‌های کانستراکتور هم نام با فیلدهای خود کلاس است و برای آنکه جاوا بتواند آنها را از هم تشخیص دهد، باید یا اسم‌ها را عوض کرد و یا از کلید واژه‌ی this در کانستراکتور استفاده کرد. در نهایت، تکه کد درست به شکل زیر خواهد بود:

```java
public class Problem4 {
   public static void main(String[] args) {
       person p1 = new person("Ted", "Mosby", 123456);
       p1.express();
       p1.express("Happy");
       p1.print();
   }
   static class person {
       private String FirstName;
       private String last_name;
       private int id;
       public person(String FirstName, String last_name, int id){
           this.FirstName = FirstName;
           this.last_name = last_name;
           this.id = id;
       }
       public void express(){
           System.out.println("I feel neutral");
       }
       public int express(String state){
           System.out.println("I feel " + state + " today");
           return 0;
       }
       public void print(){
           System.out.println("person{" +
                   "name='" + FirstName + '\'' +
                   ", lname='" + last_name + '\'' +
                   ", id=" + id +
                   '}');
       }
   }
}
```

سوال چهارم

- **الف)**
  - **۱.** آرایه، همانطور که از اسمش معلوم است، شامل تعدادی متغیر از یک نوع است. آرایه‌ها اندازه‌ی ثابتی دارند و هنگام تعریف شدن، اندازه‌ی آنها معلوم می‌شود. برای مقادیر داخل آرایه محدودیت خاصی وجود ندارد و تنها باید از یک نوع باشند. ArrayList را می‌توان یک نسخه‌ی قوی‌تر (و البته کندتر) از آرایه دانست. ArrayList همان آرایه است، با این تفاوت که می‌توان سایز آن را تغییر داد و اندازه‌ی آن همیشه تقریباً به تعداد متغیرهای درون آن است. برای ArrayList علاوه بر این محدودیت که نوع‌های متغیرهای درون آن باید یکی باشند وجود دارد، این محدودیت هم هست که نوع متغیرهای ذخیره شده‌ی آن، باید حتماً یک شئ از یک کلاس باشند و نمی‌توان نوع‌های ابتدایی را در آن ذخیره کرد. LinkedList اما ساختاری است که می‌توان آن را قوی‌تر از ArrayList (و البته کندتر) دانست. زیرا LinkedList علاوه بر دارا بودن تمام مزیت‌های ArrayList که گفته شد، این مزیت را دارد که از هر دو طرف، آرایه را افزایش و یا کاهش دهد. در صورتی که ArrayList تنها می‌تواند از سمت راست این کار را انجام دهد. اما از طرفی، امکان دسترسی به یک عضو در وسط آرایه را ندارد و برای دسترسی، لازم است که تمام آرایه را پیمایش کند تا به آن برسد و این ویژگی، LinkedList را در دسترسی به اعضا بسیار کند می‌کند.
  - **۲.** از آنجا که لازم است که بیماران، دائم از لیست پاک شوند و یا به لیست اضافه شوند، به ساختاری نیاز داریم که این عملیات را سریع انجام دهد. از آنجا که آرایه و ArrayList، اعضای خود را به صورت یک قطعه‌ی متوالی در حافظه ذخیره می‌کنند و پس از پاک کردن یک بیمار نیز باید این ویژگی را حفظ کنند، لازم است که از مدیریت حافظه استفاده کنند و این باعث می‌شود که خیلی کند اجرا شوند. LinkedList اما، علی رغم این که دسترسی سریع به یک بیمار در وسط لیست را ندارد، اما به راحتی می‌تواند لیست را نمایش دهد و افراد را از لیست حذف و یا به آن اضافه کند. در نتیجه LinkedList بهترین گزینه در مورد خواهد بود.

- **۳.** بله، می‌توان یک شئ را چند بار به یک ArrayList اضافه کرد.
- **۴.** در جاوا تابع ()length برای آرایه تعریف نمی‌شود. در عوض، آرایه‌ها دارای فیلدی به نام length هستند که اندازه‌ی کل آرایه را نگه می‌دارد (نه تعداد اشیاء داخل آن). اما ArrayList متدی به نام ()size دارد که تعداد اشیاء داخل آن را بر می‌گرداند (نه اندازه‌ی کل لیست). برای همین، هنگام اجرای حلقه بر روی یک آرایه، باید به این نکته دقت کرد که مقدار length آن آرایه، اندازه‌ی کل آن را نشان می‌دهد که در اکثر مواقع ما نیازی به آن نداریم و صرفاً با خانه‌هایی از آن که تغییر دادیم کار داریم.