امیرضا میرزایی - ۹۸۱۰۶۱۱۲

(۱)

الف) JRE محیط اجرای برنامه جاوا می‌باشد. JRE حاوی کلاس‌ها، class loader و JVM است

ب) انتظار اسپکا یعنی نوع شی را در که تعیین کنیم یعنی کامپایلر نوع پیاده‌سازی متد را برای آن شی بداند

ج) با استفاده از این کلاس می‌توانیم لیست متدهای یک کلاس را برگردانیم و یا از طریق این کلاس متد کلاس دیگری را راز کنیم

د) پیمایشگر: یک اینترفیس است که در collection ها از آن استفاده شده است و امکان پیمایش یک collection را فراهم می‌کند

ه) سریال پذیر: یک اینترفیس خالی است که صرفاً به کامپایلر می‌گوید که کلاسی که آن را implement کرده سریال پذیر است

(۲) کلاس final کلاسی است که extend نمی‌شود

کلاس immutable کلاسی است که اجزای آن قابل تغییر نیستند (private هستند و setter ندارند)

در کل بهتر است کلاس immutable ، final باشد زیرا می‌توان با extend کردن کلاس و تعریف متغیرهای دیگر immutable بودن کلاس را خراب کرد

امیررضا میرزایی - ۹۸۱۰۵۹۱۱۲

(۳) الف) جریان اصلی برنامه در حقیقت منطق اصلی برنامه بدون در نظر گرفتن حالت خاص هست

در جایی جانبی برنامه یکسری حالت‌های خاص است که به منطق اصلی ما ربطی ندارند

ب) در java با استفاده از Expection می‌توانیم روند جانبی را از روند اصلی جدا کنیم

اگر از expection استفاده نکنیم مجبوریم برای هر سناریو یک if بگذاریم و این باعث

ناخوانایی در روند اصلی برنامه می‌شود و حجم کد را هم زیاد می‌کند

مزیت دیگر این است که دیگر لازم نیست حالت خاص پیش آمده را همانجا handle کنیم

و منطق اصلی را به هم بریزیم و می‌توانیم با استفاده از throw مسئولیت handle

کردن مشکل پیش آمده را به کلاس دیگری بدهیم

(۴) در لیست پیوندی هر خانه آدرس خانه‌ی بعد خود را نگه می‌دارد و خانه‌ها لزوماً پشت هم نیستند

در لیست پیوندی عملیات حذف یا اضافه کردن یک خانه به یک جای مشخص در O(1) انجام

می‌شود زیرا کافیست آدرس بعد خانه را عوض کنیم اما عملیات گرفتن یک خانه خاص و یا در کل

پیدا بیشتر برای پیدا کردن خانه‌ای خاص در O(n) انجام می‌شود

آرایه یک حافظه‌ی پشت سر هم است و عملیات گرفتن یک خانه‌ی خاص در O(1) انجام می‌شود

اما برای اضافه کردن و یا حذف کردن یک خانه باید تمام خانه‌ی راست آن یکی به راست شیفت

داده شوند پس در O(n) انجام می‌شود

امیر رضا بیرزایی ۹۸۱۰۶۱۱۲

(۵) کامپایلر با دیدن @override چک می‌کند که آیا متد در کلاس پدر وجود دارد یا نه و اگر نداشت خطای کامپایل می‌دهد

منطقی

اجباری برای استفاده از @override نیست اما بهتر است از آن استفاده کنیم تا جلوی خطا را بگیریم برای مثال ممکن است فکر کنیم متدی در کلاس پدر وجود دارد و ما در کلاس فرزند آن را override کرده‌ایم ولی در کلاس فرزند اسم متد را اشتباه نوشته‌ایم.

(۶) j استاتیک است و در سطح کلاس وجود دارد یعنی حتی بدون داشتن نمونه‌ای از کلاس j در حافظه وجود دارد و می‌توان از آن در متدهای استاتیک کلاس استفاده کرد.

اما i در هر نمونه‌ی کلاس وجود دارد یعنی تا قبل از new کردن i وجود ندارد و در هر نمونه از یک شی هم یک i وجود دارد (این دو با یکدیگر نیستند)

(۷)

```
public boolean equals(object o) {

if (o == null) return false;
if (o instance of Frait == false) return false;

Frait frait = (Frait) o;

return Double.valueOf(weight).equals(Double.valueOf(frait.weight))
    && name.equals(frait.name) && color.equals(frait.color);

}
```

امیرضا میرزایی 9۸۱۰۶۱۱۲

۸- مهم‌ترین مشکل پیش آمده این نیست که باید هر خطا را در همانجایی که رخ می‌دهد برایش رفع کرد
معمولاً خطا را کلاسی که یا متدی که از متد خطادار استفاده می‌کند رفع می‌کند اما بدون
وجود این Framework مجبوریم خطا را همانجا رفع کنیم و این باعث مخلوط
شدن روند اصلی و روند کاپی برنامه می‌شود

۹-

A) اگر queue پر بود متد اینجا متوقف می‌شود تا وقتی در putElement ، notify شود

B) این notify برای wait() خط ۲۷ است و می‌گوید که دیگر queue خالی
نیست و عملیات get می‌تواند انجام شود

C) این notify برای A است و می‌گوید که دیگر queue پر نیست و عملیات
put می‌تواند انجام شود

D) اگر queue خالی بود تمام ترد‌هایی که در (emptyQueue) synchronised لاک شده‌اند باز می‌شوند و به کارشان ادامه
می‌دهند

```
while (producer.hasNext()) {
try {
producer.getElement(); }
catch (InterruptedExpection expection) { }
}
```