Refactoring

بسته‌ی Refactoring را دانلود کنید.

این بسته شامل سه فایل *java* یعنی *First.java* و *Second.java* و *Third.java* است.

فایل‌ها را با توجه به نکات زیر *Refactor* کنید:

- در ابتدای هر فایل یک عدد به صورت کامنت نوشته‌شده است که حداقل تعداد *Bad smell*هایی است که باید در آن‌ها پیدا کنید.
- در ابتدای فایل‌های *Refactor* شده با فرمت زیر هریک از *Bad smell*های پیداشده و روش رفع آن‌ها را بنویسید:

```
#number    badSmell    solution
```

- کد موجود در هر فایل به درستی کار میکند بنابراین اگر پس از انجام *Refactoring* همچنان درست کار نکنند نمره‌ای تعلق ***نخواهد گرفت***. **دقت کنید که کد شما باید تمامی تست‌ها را پاس کند و صرفا پاس‌کردن برخی تست‌ها که یکسری بررسی‌های ساده را انجام می‌دهند، نمره‌ای ندارد.**
- نوشتن مستند برای کلاس‌ها و متدهای هر فایل *Refactor* شده به صورت *javadoc* ، با رعایت ارف و استفاده‌ی به‌جا از *tag*های آن نمره‌ی امتیازی دارد.

آن چه که باید آپلود کنید

یک فایل *zip* است که فقط شامل فایل‌های *Refactor* شده‌ی *First.java* و *Second.java* و *Third.java* است. در فایل‌ها نام کلاس‌ها را عوض **نکنید**.

ArrayList

فایل SimpleArrayListTest را دانلود کنید.

کلاس SimpleArrayList شامل متدهای با پروتوتایپ زیر را به گونه‌ای پیاده‌سازی کنید که تست‌های فایل دانلود شده را پاس کند.

```
1  public int size()
2  public boolean isEmpty()
3  public void clear()
4  public Integer get(int index)
5  public Integer set(int index, Integer element)
6  public boolean add(Integer element)
7  public void add(int index, Integer element)
8  public boolean addAll(Integer... elements)
9  public Integer remove(int index)
```

- فایل دانلود شده را به دقت مطالعه کنید تا با عملکرد متدها و پارامترها، انواع برگشتی و اکسپشن‌های پرتاب شده توسط آن‌ها آشنا شوید.
- درصورتی که همچنان در پیاده سازی متدها ابهام داشتید، پیشنهاد می‌شود مستندات مربوط به آن‌ها را در واسط java.util.List مطالعه نمایید.
- اگر با نحوه‌ی پیاده سازی آرایه با طول متغیر نیز آشنایی ندارید، پیشنهاد می‌شود مستندات کلاس java.util.ArrayList را بخوانید (یا اینکه به سادگی آن را *search* کنید!!!).
- دقت کنید که حق ارث‌بری یا استفاده از کلاس‌های List آماده ندارید و نمره شما **صفر** می‌شود.

آن چه که باید آپلود کنید

یک فایل *zip* که در آن **فقط** فایل SimpleArrayList.java وجود داشته باشد.

شرکت بیمه

شرکت بیمه تصمیم دارد برای به دست آوردن اطلاعات مشتریان خود از یک برنامه‌ی جاوا استفاده کند. با توجه به اینکه کیفیت این نرم افزار بسیار اهمیت دارد مدیر شرکت در نظر دارد که نحوه‌ی تولید آن مبتنی بر روش TDD باشد. به این منظور علاوه بر پروتوتایپ و توضیح عملکرد متدهای موردنیاز، کلاس‌ها اصلی ذخیره‌سازی افراد را در قالب فایل یک بسته در اختیار مهندسان قرار می‌دهد تا بتوانند تست‌های نرم افزار را به خوبی تولید کنند.
نوشتن واحدهای تست نرم‌افزار کلاس کمکی InfoExtractor به عهده شماست. پس با توجه به نکات زیر آن‌ها را بنویسید.

ابتدا فایل OPP را دانلود کرده و به آن نگاه کنید.

در هر بارگذاری کلاس، لازم است متد زیر فراخوانی شود تا بیشینه‌ی محاسبات متدها بر اساس آن مشخص شود.

```
1 | public static void setMax(int max)
```

- تمامی متدها در صورت خالی بودن لیست ورودی یا null بودن آن، خطای *EmptyInputException* پرتاب می‌کنند.
- تمامی متدها در صورت بیشتر بودن اندازه‌ی لیست ورودی از بیشینه‌ی حد محاسباتی، خطای *DataInputOverflowException* را پرتاب می‌کنند.
- تمامی متدها در صورت وجود عنصر null در لیست ورودی خطای *NullPointerException* پرتاب میکنند.

کلاس شامل چهار متد زیر می‌باشد:

```
1 | public static List<Integer> getPaymentsPerHour(List<Person> people)
```

خروجی متد، لیستی از مقدار **کف** حقوق هر فرد در ساعت است. (یک سال را ۳۶۶ روز در نظر بگیرید)

فرمول زیر را برای محاسبه‌ی حقوق هر فرد در ساعت استفاده کنید:

$$\lfloor \frac{PaymentPerMonth \times 12}{366 \times WorkHoursPerDay} \rfloor$$

```
public static Person getLaziest(List<Person> people)
```

در گروهی از افراد، تنبل‌ترین فرد شخصی بیکار است (نمونه‌ای از کلاس *Employee* نیست) که نسبت به بقیه افراد بیکار آن گروه، موجودی حساب کمتری دارد.
متد در صورت وجود نداشتن فرد بیکار در لیست ورودی null و در صورت برابری موجودی حساب میان تنبل‌ترین افراد، نزدیک‌ترین آن‌ها به ابتدای لیست ورودی را برمیگرداند.

```
public static List<Employee> sortOnSuccess(List<Person> people)
```

خروجی متد، لیستی از **کارمندان** موجود در لیست ورودی می‌باشد که براساس موفقیت به صورت صعودی مرتب شده است. از میان دو فرد، شخص موفق‌تر در زمان کمتر حقوق بیشتری دریافت می‌کند، در صورت برابری این شاخص بین دو فرد، فرد با سن کمتر موفق‌تر است. دو فردی که حقوق مساوی در زمان دریافت می‌کنند و هم‌سن هستند، باید در لیست برگردانده شده ترتیب اولیه را نسبت به یکدیگر داشته باشند (به عبارتی اگر شخص اول در لیست ورودی به ابتدای لیست نزدیک‌تر است، باید در لیست خروجی هم نسبت به شخص دوم به ابتدای لیست نزدیک‌تر باشد).

```
1 | public static List<Person> getRichestSorted(List<Person> people)
```

خروجی متد، لیستی از افراد ثروتمند موجود در لیست ورودی می‌باشد که براساس نام به صورت صعودی مرتب شده است. در گروهی از افراد، شخصی ثروتمند است که حقوق و موجودی حساب او از میانگین گروه بیشتر باشد. مشابه متد قبل، دو فرد با اسم برابر باید با ترتیب اولیه نسبت به یکدیگر در لیست برگردانده شده قرار گیرند.

- برای هر متد بالا، باید یک تست با ورودی‌های متفاوت و بررسی تمام اکسپشن‌های پرتاب شده، نوشته شود.
- وجود کد تکراری در تست‌ها برای مقداردهی اولیه، عملیات نهایی و ... **نمره‌ی منفی** دارد. برای جلوگیری، از امکانات *junit* استفاده کنید.
- پس از نوشتن تست‌ها، کلاس *InfoExtractor* را پیاده‌سازی کنید. تست‌های آن را در محیط توسعه‌ی نرم‌افزار خود اجرا کرده و با استفاده از امکانات آن، *code coverage* را مشاهده کنید.
- حداقل *"code coverage"* برای دریافت نمره کامل، 85% می‌باشد.
- کلاس *InfoExtractor* پیاده‌سازی شده باید درست کار کند، در غیر این صورت تست‌ها و *code coverage* آن بررسی نخواهد شد و نمره‌ای به آن تعلق **نخواهد گرفت**.

آن چه که باید آپلود کنید

یک فایل zip که فقط شامل فایل‌های InfoExtractor.java و TestInfoExtractor.java می‌باشد.

تعادل در پردازش (امتیازی)

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

در سایت دانشکده مهندسی کامپیوتر، n سرور برای انجام *"task"*های کامپیوتری وجود دارد. *تعداد عملیات مربوط به هر سرور مشخص است اما جهت افزایش سرعت و کارایی لازم است آن‌ها را تا حد امکان بین سرورها پخش کرد. به عبارتی، اختلاف بین تعداد عملیات سرور با بیشترین *task و سرور با کمترین *task* باید به حداقل برسد.
با فرض اینکه منتقل کردن یک *task* از سرور به سرور دیگر، **یک** ثانیه زمان ببرد، برنامه‌ای بنویسید که حداقل زمان لازم را برحسب ثانیه، برای ایجاد تعادل پردازشی بین سرورها محاسبه کند.

ورودی

در خط اول ورودی n یا تعداد سرورها داده می‌شود.
خط بعدی شامل دنباله$m_1, m_2, ..., m_n$ است که در آن m_i نشان دهنده‌ی تعداد عملیات سرور *i*ام است.

$$1 \leq n \leq 10^4$$

$$0 \leq m_i \leq 2 * 10^4$$

خروجی

حداقل تعداد ثانیه‌های لازم را برای ایجاد تعادل پردازشی میان سرورها چاپ کنید.

مثال

ورودی نمونه ۱

```
2
1 6
```

خروجی نمونه ۱

```
2
```

دو *task* از سرور دوم به سرور اول باید منتقل شود.

ورودی نمونه ۲

```
7
10 11 10 11 10 11 11
```

خروجی نمونه ۲

```
0
```

ورودی نمونه ۳

```
5
1 2 3 4 5
```

خروجی نمونه ۳

```
3
```

در یکی از حالت‌ها به ترتیب سه *task* باید منتقل شوند:

از سرور چهارم به اول بنابراین دنباله عملیات سرورها به صورت $2, 2, 3, 3, 5$ خواهد بود.

از سرور پنجم به اول بنابراین دنباله عملیات سرورها به صورت $3, 2, 3, 3, 4$ خواهد بود.

از سرور پنجم به دوم بنابراین دنباله عملیات سرورها به صورت $3, 3, 3, 3, 3$ خواهد بود.

دنباله پرانتزی (امتیازی)

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت

دنباله پرانتزی، رشته‌ای شامل کاراکترهای '**(**' و '**)**' است. دنباله پرانتزی زمانی **منظم** است که پرانتزهای آن بتوانند در عبارت ریاضی درست استفاده شوند. به عنوان مثال دنباله‌های "()()" و "((()))" منظم هستند اما ")(" و "(" نه. بین دو دنباله پرانتزی عملگر *concatenation* را به صورت **+** تعریف می‌کنیم، به عبارتی $s + t$ رشته حاصل از چسباندن دنباله پرانتزی t به سمت راست دنباله پرانتزی s را نشان می‌دهد.

برنامه‌ای بنویسید که با گرفتن n دنباله پرانتزی به صورت $s_1, s_2, ..., s_n$ تعداد جفت‌های $i, j (1 \leq i, j \leq n)$ را پیدا کند به طوری که دنباله پرانتزی $s_i + s_j$ منظم باشد. توجه شود که اگر $s_i + s_j$ و $s_j + s_i$ هر دو منظم باشند به طوری که $i \neq j$، باید هر دو حالت را به جواب اضافه کرد، همچنین منظم بودن دنباله پرانتزی $s_i + s_i$ یکی به جواب اضافه می‌کند.

ورودی

در خط اول ورودی n یا تعداد دنباله‌های پرانتزی دریافت شده داده می‌شود و n خط بعدی شامل دنباله‌های پرانتزی می‌باشد. تضمین می‌شود که رشته مربوط به هر یک از دنباله‌های پرانتزی خالی نیست و مجموع طول آن‌ها یا s در بازه زیر قرار می‌گیرد.

$$1 \leq n, s \leq 3 * 10^5$$

خروجی

در یک خط تعداد جفت‌های $i, j (1 \leq i, j \leq n)$ به صورتی که دنباله پرانتزی حاصل از $s_i + s_j$ منظم باشد را چاپ کنید.

مثال

ورودی نمونه ۱

```
3
)
()
(
```

خروجی نمونه ۱

```
2
```

جفت‌های $(3, 1)$ و $(2, 2)$ دنباله پرانتزی منظم ایجاد می‌کنند.

ورودی نمونه ۲

```
2
()
```

( )

خروجی نمونه ۲

4

جفت‌های $(2,2)$ ، $(1,2)$ ، $(2,1)$ و $(1,1)$ دنباله پرانتزی منظم ایجاد می‌کنند.