کَسر!

در این سؤال، شما قرار است یک کلاس طرّاحی کنید که برای نگه‌داریِ اعدادِ کسری استفاده می‌شود. مشخّصاتِ این کلاس به شرحِ زیر است:

Data Members

- صورت و مخرج: صورتِ کسر یک عددِ حسابی (صفر + اعدادِ طبیعی) و مخرجِ کسر یک عددِ طبیعی است. **این دو عدد را روی heap تعریف کنید.**
- علامت: علامتِ کسر (مثبت یا منفی) در این متغیّر ذخیره می‌شود.

Methods

- تابعِ سازنده (constructor): این کلاس سه تابعِ سازنده دارد. تابعِ اوّل دو ورودی می‌گیرد که شاملِ صورت و مخرج است. تابعِ دوم یک ورودی می‌گیرد که برابر با مقدارِ صحیحِ کسر است (مثلاً ۲. در این حالت صورت ۲ و مخرج ۱ است). تابعِ سوم ورودی‌ای ندارد و با این کار، مقدارِ کسر باید برابر با صفر باشد.
- تابعِ simplify: با صدا زدنِ این تابع روی یک object، کسرِ ذخیره‌شده باید ساده شود! این تابع، هم objectای که این تابع روی آن صدازده‌شده را عوض می‌کند، و هم یک reference را به آن برمی‌گرداند. به عنوانِ مثال به کدِ زیر توجّه کنید (f1 و f2 هر دو کسر هستند):

```
1 | f2 = f1.simplify();
```

کدِ زیر مقدارِ f1 را تغییر می‌دهد، و همچنین object حاصل‌شده را در f2 می‌ریزد.

- تابعِ inverse: با صدا زدنِ این تابع روی یک object، کسرِ ذخیره‌شده باید مقلوب شود (جای صورت و مخرج عوض شود). مثلِ تابعِ قبلی، در این تابع نیز هم object مربوطه را تغییر دهید و هم یک reference به object تغییرداده‌شده برگردانید.

- تابعِ count: این تابع باید تعدادِ کسرهایی که تابه‌حال ساخته شده است را برگرداند. همچنین این تابع باید قابلیّتِ این را داشته باشد که بدونِ کمکِ یک object بتوانیم صدایش بزنیم.
- همچنین برای هر سه Data Member توابعِ getter و setter تعریف کنید.

Overloaded operators

- عمل‌گرِ جمع (+): عمل‌گرِ جمع را طوری پیاده‌سازی کنید که عبارتِ زیر معتبر باشد (هر سه متغیّرِ استفاده‌شده کسر هستند):

```
f1 + f2 + f3;
```

دقّت کنید که عمل‌گرِ جمع طبیعتاً نباید روی عمل‌وندهایش تأثیر بگذارد و صرفاً باید جمعِ دو کسر را برگرداند. همچنین در این مرحله نیازی به ساده‌کردنِ کسر نیست.

- عمل‌گرِ انتساب (=): عمل‌گرِ انتساب را طوری پیاده‌سازی کنید که عبارتِ زیر معتبر باشد (هر سه متغیّرِ استفاده‌شده کسر هستند):

```
f1 = f2 = f3;
```

- عمل‌گرِ تساوی (==): عمل‌گرِ تساوی باید مساوی بودنِ مقدارِ کسرها را نشان دهد.
- عمل‌گرِ >>: این عمل‌گر را طوری پیاده‌سازی کنید که عبارتِ زیر معتبر باشد (f1 و f2 هر دو کسر هستند):

```
std::cin >> f1 >> f2;
```

- عمل‌گرِ <<: این عمل‌گر را طوری پیاده‌سازی کنید که عبارتِ زیر معتبر باشد (f1 و f2 هر دو کسر هستند):

```
std::cout << f1 << f2 << std::endl;
```

نکاتِ پایانی

- برنامه‌ی شما نباید memory leak داشته باشد. یعنی از اشغال‌کردنِ حافظه‌ی اضافی اجتناب کنید.
- کدِ شما باید به ازای کدِ زیر درست کار کند:

```
void inc( Fraction f ) {
    Fraction temp(5);
    f = f + temp;
}

int main () {
    Fraction f1(15, 4 );
    inc( f1 );
}
```

منظور از «درست کار کردن» این است که دقّت کنید که تابعِ inc یک کپی از f1 را ورودی بگیرد و نتواند مقادیرِ موجود در f1 را تغییر دهد. راهنمایی: copy constructor!

- هرجا که در محاسبات به مشکل برخوردید (مثلاً مخرجِ صفر)، یک پیامِ مناسب به کاربر نشان دهید و محاسباتِ مربوطه را انجام ندهید. به عنوانِ مثال در تابعِ مقلوب کردن، ممکن است مخرجِ کسر صفر شود. بنابراین یک ارور چاپ کنید و کسر را مقلوب نکنید.

اقتصادی ترین

کد زیر قسمتی از یک برنامه کامل است که می تواند کمترین قیمت را از بین تعدادی خودرو پیدا کند و در متغیر least_price قرار دهد. بدون پیاده سازی کلاس Car کد زیر را با استفاده از مفهوم stream api طبق جاوا 8 بازطراحی کنید و در نهایت متغیر least_price را چاپ کنید.

نکته:

- فرض کنید لیست cars خالی نیست و تعداد مشخصی (به طور مثال 10) عضو دارد.
- کلاس car رو کلا کاری نداشته باشید و نه پیاده‌کنید نه هیچی. فرض کنید هست.

```
int least_price = 50;
ArrayList<Car> cars = new ArrayList<>();
for(Car car : cars){
    if(car.owned() == true){
            if(car.price() < least_price)
                least_price = car.price();
    }
}
```

جاوا ۷۸

کدِ زیر با نسخه‌ی ۷ جاوا نوشته شده است. آن را با جاوا ۸ دوباره پیاده‌سازی کنید. نیازی به طرّاحیِ کلاسِ Human نیست.

- نکته: از مفهومِ default method interface استفاده کنید.
- پیاده‌سازی اینترفیس طبیعتا نباید به پیاده‌سازی HUMANوابسته باشد و داخلش از هیچ متدی از human استفاده نکنید.

```
interface Java7Interface {
    boolean isInfected( Human human );
    boolean isHealthy( Human human );
}
public class Java7 implements Java7Interface {
    public boolean isInfected( Human human ) {
        return human.isInfected()
    }
    public boolean isHealthy( Human human ) {
        return !( human.isInfected() );
    }
}
```