ساعت شطرنج

- نام طراح: میثم باوی
- سطح سؤال: متوسط

در این سؤال قصد داریم به کمک الگوی طراحی استراتژی، یک ساعت شطرنج طراحی کنیم که دارای سه حالت مختلف است. در مورد این الگو مطالعه کنید.

پروژهٔ اولیه را از این لینک دانلود کنید.

قوانین کلی

- در شطرنج دو بازیکن با مهره های سفید و سیاه داریم. نوبت شروع بازی همیشه با بازیکن سفید است.
- در ابتدا به هر دو بازیکن زمان برابر داده می‌شود.
- زمانی که هر بازیکن در نوبت خود برای فکر کردن صرف می‌کند از زمان او کم می‌شود.
- اگر زمان بازیکنی به صفر برسد، او بازنده است و زمان ها تا راه اندازی مجدد ساعت دیگر تغییری نمی‌کنند (زمان هیچ گاه منفی نمی‌شود).
- هنگامی که یک بازیکن حرکتش را انجام می‌دهد دکمهٔ **مخصوص خودش** را فشار می‌دهد تا نوبت حریف شود.
- همیشه تنها دکمهٔ بازیکنی که نوبت اوست فعال است و فشار دادن دکمهٔ بازیکن دیگر اثری ندارد.

کلاس ChessClock

- سازندهٔ این کلاس یک شئ از رابط `ClockStrategy` دریافت می‌کند که تعیین کنندهٔ نحوهٔ عملکرد ساعت است. پارامتر دیگر `startingTimeMillis` است که از نوع `long` و مقدار زمان هر بازیکن در شروع به میلی‌ثانیه است.
- متد `passTime` به برنامه می‌گوید که زمان سپری شده است. پارامتر این متد مقدار زمان سپری شده به میلی‌ثانیه است. با فراخوانی پی در پی این متد، زمان های سپری شده جمع می‌شوند.

- متد های `getWhiteTime` و `getBlackTime` میزان زمان کنونی بازیکن با مهره های سفید و سیاه را برمی‌گردانند.
- متد `isWhiteTurn` اگر نوبت بازیکن سفید باشد `true` و اگر نوبت بازیکن سیاه باشد `false` برمی‌گرداند.
- متد `isReachedZero` اگر زمان یکی از بازیکن ها صفر شده باشد `true` بر می‌گرداند و نشانهٔ پایان بازی است، در غیر این صورت `false` بر می‌گرداند.
- متد های `onWhitePress` و `onBlackPress` نشانهٔ فشرده شدن دکمهٔ مربوط آن بازیکن است و **در صورت نیاز**، نوبت تغییر می‌کند.
- متد `changeSettingAndReset` یک استراتژی جدید و یک زمان اولیهٔ جدید دریافت می‌کند و ساعت reset می‌شود؛ یعنی زمان بازیکن ها برابر با زمان اولیهٔ جدید می‌شود، نوبت به بازیکن سفید می‌رسد و عملکرد ساعت از این به بعد با استراتژی جدید انجام می‌شود.

کلاس ClockState

این کلاس نگهدارندهٔ اکثر متغیر های لازم برای پیاده سازی ساعت (مثلاً متغیر زمان بازیکن ها) و بیانگر وضعیت ساعت است. ورودی متد های رابط `ClockStrategy` شیئی از این کلاس خواهند بود تا تغییرات لازم روی همین شئ اعمال شود.
جزئیات پیاده سازی این کلاس به عهدهٔ خودتان است و مستقیماً مورد تست قرار نمی‌گیرد.

رابط ClockStrategy

- متد `updateTimes` با دریافت وضعیت ساعت و با توجه به زمان سپری شده از آخرین آپدیت، زمان بازیکن ها را به روز می‌کند. این متد نوبت را تغییر نمی‌دهد.
- متد `updateTimesAndChangeTurn` بیانگر فشرده شدن دکمهٔ ساعت و تغییر نوبت است؛ ابتدا زمان‌ها به روز می‌شوند و سپس نوبت تغییر می‌کند و کار های مربوط به لحظهٔ تغییر نوبت انجام می‌شود.

کلاس NormalIncrementStrategy

- این کلاس رابط `ClockStrategy` را پیاده سازی می‌کند.
- در این حالت، به زمان بازیکن **پس** از انجام حرکتش مقدار `increment` اضافه می‌شود.

- مقدار `increment` از نوع `long` است و در سازندهٔ این کلاس دریافت می‌شود.

کلاس BronsteinIncrementStrategy

- این کلاس رابط `ClockStrategy` را پیاده سازی می‌کند.
- در این حالت مانند NormalIncrement، به زمان بازیکن **پس** از انجام حرکتش مقداری زمان اضافه می‌شود. با این تفاوت که تنها بخشی از `increment` به او اضافه می‌شود که از آن استفاده کرده؛ مثلا اگر مقدار `increment` برابر 10 است و حرکت بازیکن تنها 5 واحد طول کشیده، به زمان او تنها همان 5 واحد اضافه می‌شود. ولی اگر بیش از 10 طول کشیده بود، 10 واحد اضافه می‌شد.
- مقدار `increment` از نوع `long` است و در سازندهٔ این کلاس دریافت می‌شود.

کلاس HourglassStrategy

- این کلاس رابط `ClockStrategy` را پیاده سازی می‌کند.
- این حالت مانند ساعت شنی عمل می‌کند؛ در حین کم شدن زمان یک بازیکن به زمان حریف او اضافه می‌شود.
- مجموع زمان بازیکن ها همیشه دو برابر `startingTimeMillis` و ثابت است.
- سازندهٔ این کلاس پارامتری ندارد.

نمونه

کد

```
ChessClock clock = new ChessClock(new NormalIncrementStrategy(3), 100);

System.out.printf("Is white turn: %b\n", clock.isWhiteTurn());

clock.passTime(25);
System.out.println();
System.out.printf("White time: %d\n", clock.getWhiteTime());
System.out.printf("Black time: %d\n", clock.getBlackTime());

clock.onWhitePress();
System.out.println("Clock pressed");
```

```
System.out.printf("White time: %d\n", clock.getWhiteTime());
System.out.printf("Black time: %d\n", clock.getBlackTime());
System.out.printf("Is white turn: %b\n", clock.isWhiteTurn());

clock.passTime(30);
System.out.println();
System.out.printf("White time: %d\n", clock.getWhiteTime());
System.out.printf("Black time: %d\n", clock.getBlackTime());

clock.onBlackPress();
System.out.println("Clock pressed");
System.out.printf("White time: %d\n", clock.getWhiteTime());
System.out.printf("Black time: %d\n", clock.getBlackTime());
System.out.printf("Is white turn: %b\n", clock.isWhiteTurn());
```

خروجی

```
Is white turn: true

White time: 75
Black time: 100
Clock pressed
White time: 78
Black time: 100
Is white turn: false

White time: 78
Black time: 70
Clock pressed
White time: 78
Black time: 73
Is white turn: true
```

نکات

- تمام مقادیر مربوط به زمان از نوع `long` هستند.
- تمام ورودی های مربوط به زمان غیر منفی هستند.
- دقت کنید که ممکن است متد `passTime` را چند بار پشت سر هم بدون فراخوانی متد دیگری فراخوانی کنیم.

- کد کلاس `ChessClock` **باید مستقل از نوع استراتژی باشد؛** یعنی برای طراحی استراتژی جدیدی به جز استراتژی های بالا، لازم نباشد هیچ تغییری در کد `ChessClock` بدهیم. تمام منطق یک استراتژی باید درون کلاس آن باشد.

آنچه باید آپلود کنید

پنج فایل `NormalIncrementStrategy.java` `ClockState.java` `ChessClock.java` `BronsteinIncrementStrategy.java` و `HourglassStrategy.java` را بدون هیچ گونه پوشه‌ای آپلود کنید.

اعداد طبیعی غیرطبیعی

- محدودیت زمان: ۱ ثانیه
- محدودیت حافظه: ۲۵۶ مگابایت
- نام طراح: مهرسا عرب زاده
- سطح سوال: متوسط

در مجموعه اعداد طبیعی، اعدادی وجود دارند که ویژگی های نسبتا خاصی دارند. برخی از آن ها نیز در داشتن آن ویژگی های خاص با یکدیگر مشترکند. در این سوال قصد داریم با استفاده از امکانات جدید جاوا، این اعداد را تشخیص دهیم و اعداد با ویژگی های مشترک را نیز در یک گروه قرار دهیم.

ابتدا فایل اولیه را از این لینک دانلود کنید.

کلاس NaturalNumbers

این کلاس دارای فیلد nums است که آرایه ای از جنس int می باشد.

در constructor این کلاس، به کمک stream، آرایه ی nums را با اعداد طبیعی 1 تا 100 مقداردهی اولیه کنید.

متدهای زیر را به شکل گفته شده پیاده سازی کنید:

1.

```
public Function<Integer, Integer> oddOrEven() {
}
```

در این متد باقی مانده ی تقسیم یک عدد بر دو محاسبه می شود.

2.

```
public String oddOrEven(int number) {
}
```

در این متد با استفاده از متد قبل زوج یا فرد بودن یک عدد مشخص می شود. در صورت زوج بودن عدد عبارت `Even` و در صورت فرد بودن آن `Odd` برگردانده می شود.

3.

```
public Predicate<Integer> isPalindrome() {
}
```

در این متد پالیندروم بودن یا نبودن یک عدد بررسی می شود.

4.

```
public String isPalindrome(int number) {
}
```

در این متد با استفاده از متد قبل، اگر عدد پالیندروم باشد عبارت `Palindrome` و در غیر این صورت عبارت `Not Palindrome` بازگردانده می شود.

5.

```
public Predicate<Integer> isPrime() {
}
```

در این متد اول بودن یا نبودن یک عدد بررسی می شود.

6.

```
public String isPrime(int number) {
}
```

در این متد با استفاده از متد قبل، در صورت اول بون عدد عبارت `Prime` و در غیر این صورت عبارت `Not Prime` برگردانده می شود.

7.

```
public Comparator<Integer> comparator() {
}
```

در این متد یک Comparator برای اعداد صحیح ساخته می شود.

8.

```
public BiFunction<Integer, Integer, Integer> compare() {
}
```

در این متد با استفاده از comparator ساخته شده در متد قبل، دو عدد صحیح باهم مقایسه می شوند.

9.

```
public String compare(int a, int b) {
}
```

در این متد با استفاده از متد قبل، دو عدد صحیح باهم مقایسه می شوند و حاصل این مقایسه ها به صورت یکی از عبارت های زیر برگردانده می شود:

```
a is less than b

a is equal to b

a is greater than b
```

10.

```
public int[] collectOdds() {
    // odd numbers
}

public int[] collectEvens() {
    // even numbers
}

public int[] collectPalindromes() {
```

```
    // palindrome numbers
}

public int[] collectPrimes(){
    // prime numbers
}

public int[] collectCompounds() {
    // not prime numbers
}
```

در متدهای بالا، به کمک stream و توابع تعریف شده در مراحل قبل، آرایه ای از اعداد دارای ویژگی خواسته شده برگردانده می شود.

11.

```
public int sum(int start, int end) {
}
```

در این متد مجموع اعداد موجود در آرایه از عضو startام تا endام به کمک stream محاسبه و برگردانده می شود.

12.

```
public int[] lessThan(int number) {
}
```

در این متد آرایه ای از اعداد کوچک تر از number با کمک stream و متدهای تعریف شده در مراحل قبل برگردانده می شود.

13.

```
public int[] greaterThan(int number) {
}
```

در این متد آرایه ای از اعداد بزرگ تر از number با کمک stream و متدهای تعریف شده در مراحل قبل برگردانده می شود.

14.

```
public int weirdOperation() {
}
```

در این متد ابتدا عضوهای تکراری آرایه را حذف کرده و سپس اعداد فرد پالیندروم را جدا می کنیم. در نهایت تعداد اعضای آرایه ی حاصل را با ماکزیمم عدد آن جمع کرده و برمیگردانیم.

15.

```
public String asciiCode() {
}
```

در این متد، ابتدا آرایه را مرتب کرده، سپس اعداد زوج آن را جدا کرده و *reverse* می‌کنیم، سپس کد *ASCII* رقم سمت چپ نوع String اعداد فرد آرایه ی حاصل را در یک String پشت هم قرار داده و برمی‌گردانیم. مثال:

۱. فرض کنید آرایه به‌صورت زیر است:

```
[21, 23, 24, 26, 25, 27, 32, 31, 30, 29, 28, 22]
```

۲. پس از مرتب‌سازی آرایه، مقادیر موجود در آرایه به‌صورت زیر خواهند بود:

```
[21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]
```

۳. اعداد زوج موجود در آرایه:

```
[22, 24, 26, 28, 30, 32]
```

۳. پس از *reverse* کردن اعداد زوج، مقادیر موجود در آرایه به‌صورت زیر خواهند بود:

```
[22, 42, 62, 82, 3, 23]
```

۴. اعداد فرد موجود در آرایه به‌صورت زیر خواهند بود:

```
[3, 23]
```

۵. کد *ASCII* رقم سمت چپ معادل رشته‌ای اعداد فوق به‌صورت زیر خواهد بود:

```
[51, 50]
```

۶. نتیجه برابر خواهد بود با:

```
5150
```

آنچه باید آپلود کنید

فایل ابتدایی که دانلود کردید را پس از کامل کردن زیپ کرده و آپلود کنید.

فراخوانی از راه دور

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

به دلیل شیوع کرونا، برنامه نویسان به این فکر افتادند که برنامه نویسی تماسی را با برنامه نویسی از راه دور جایگزین کنند.

در این شیوه برنامه نویسی، بقیه برنامه نویسان با دادن دستور به سه متد واسط، از برنامه می‌خواهند شیء مورد نظر ساخته شود، متد مورد نظر call شود یا ... .

پس از شما خواسته‌اند برنامه‌ای با استفاده از رفلکشن پیاده سازی کنید که این عمل را انجام دهد.

دستوراتی که باید برنامه شما بتواند هندل کند به شرح زیر است:

برای فهم بیشتر مثال هایی که در ادامه خواهد آمد از کلاس فرضی زیر استفاده خواهد شد:

```
class Person{
    int age;
    String name;
    public Person(String name,int age){
        this.name = name;
        this.age = age;
    }
    public Person(String name){
        this.name = name;
        age = 18;
    }
    public Person(int age){
        this.age = age;
        name = "anonymous";
    }
    public void sayHello(int count, String secondName){
        for(int i = 0; i < count; i++){
            System.out.println("%s: Hello %s.\n");
        }
    }
}
```

پیش از شروع **حتما** فایل اولیه سوال را از این لینک دانلود کنید. تغییر امضای توابع باعث مشکل در داوری خواهد شد.

۱. ساخت آبجکت

متد `creator` در فایل نمونه را طوری پیاده‌سازی کنید که با دریافت یک آرگومان رشته‌ای که حاوی پیامی با فرمت زیر است، **عملیات ساخت شیء** را انجام دهد.

فرمت پارامتر `String command` :

```
create class_name arg1_class_name constructor_arg1, ..., argN_class_name constructo
```

با وارد شدن دستور `create` و دریافت `class_name` باید آبجکتی از کلاس مورد نظر ساخته شود و **در انتها شی ساخته شده برگشت داده شود.** همچنین برای پاس دادن آرگومان های مختلف به سازنده این کلاس، در ادامه، به صورت جفت جفت ابتدا نوع آرگومان و سپس مقدار آرگومان به ترتیب پارامتر های سازنده کلاس مورد نظر وارد می‌شود.

برای سادگی نوع آرگومان وارد شده **فقط** یکی از مقادیر زیر خواهدبود:

۱. `int`
۲. `double`
۳. `String`

مثالی از کاربرد این متد:

```
1 | Person p = (Person) RemoteCaller.creator("create ir.sbu.Person String mahmood
```

شیئی از کلاس `Person` از پکیج `ir.sbu` با استفاده از کانستراکتور تک پارامتری این کلاس و پاس دادن آرگومان مورد نیاز سازنده ساخته و ریترن می‌شود.

۲. صداکردن متد

متد `caller` در فایل نمونه را طوری پیاده‌سازی کنید که با دریافت دو آرگومان به ترتیب رشته ای با فرمت زیر و یک آبجکت، **عملیات صداکردن متد** را **روی آبجکت پاس داده شده** انجام دهد.

```
call method_name arg1_class_name method_arg1, ..., arg2_class_name method_arg2
```

با وارد شدن دستور `call` از شیئی که به عنوان پارامتر داده شده (`baseObject`) متدی با نام `method_name` و در صورتی که متد پارامتر خواست، در ادامه پارامتر هایی مشابه حالت پارامترهای سازنده می‌آید.

برای سادگی نوع آرگومان وارد شده **فقط** یکی از مقادیر زیر خواهدبود:

1. `int`
2. `double`
3. `String`

مثالی از کاربرد این متد:

```
1 | RemoteCaller.caller("call sayHello int 2, String ahmad", p);
```

متد `sayHello` از شیء `p` (که در مثال قبل از متد `createor` دریافت کردیم)با آرگومان های اینتجری `۳` و استرینگی `ahmad` کال می‌شود. که با توجه به بدنه این متد، با خروجی زیر مواجه می‌شویم.

```
mahmood: Hello ahmad
mahmood: Hello ahmad
```

۳. مقداردهی فیلدها

متد `setter` در فایل نمونه را طوری پیاده‌سازی کنید که با دریافت دو آرگومان به ترتیب رشته ای با فرمت زیر و یک آبجکت، **عملیات ست کردن مقدار روی یک فیلد** را **روی آبجکت پاس داده شده** انجام دهد.

```
set field_name value_class_name new_value
```

با وارد شدن دستور `set` شیئی از نوع `value_class_name` با مقدار `new_value` ساخته و روی فیلد `field_name` ذخیره سازی می‌شود.

مثال:

```
RemoteCaller.setter("set age int 19", p);
```

فیلد `age` در شیئی که با دستور اول ساخته شد با مقدار اینتیجری ۱۹ مقدار دهی می‌شود.

تضمین می‌شود:

- اگر آرگومان‌های یک کانستراکتور یا متد رشته باشد، بین رشته کاما (,) و اسپیس وجود نخواهد داشت.
- همان طور که در بدنه سوال گفته شد، تایپ آرگومان‌های سازنده و متدها در دستورات `create` ، `call` و یکی از سه مقدار `int` ، `double` یا `String` خواهد بود و متد یا سازنده ای با آرگومانی غیر از این تایپ‌ها صدا زده نخواهد شد.
- در دستور `call` متدهایی با مقدار برگشتی (غیر وید) کال خواهندشد.
- نام کلاسی که قرار است نمونه ای از آن ساخته شود به صورت کامل و با پکیج وارد می‌شود. مثلا اگر کلاس `Person` در پکیج `ir.sbu` قرار داشته باشد:

```
create ir.sbu.Person ...
```

راهنمایی

- برای فراخوانی کانستراکتور یا متد با تایپ آرگومان های وارد شده، از `primitive.class` استفاده کنید.

مثال:

```
//get constructor: Person(int age);

Constructor c = Class.forName("ir.sbu.Person").getConstructor(int.class);
```

- برای ساخت یک شی و پاس دادن به متد یا کانستراکتور نیز از `Wrapper Class` مربوط به `int` و `double` و متد `valueOf` بهره ببرید.

کلاس های `Wrapper` و کلاس `String` در پکیج: `java.lang` قرار دارند.

```
//instantiate an int or a double

Object x = Class.forName("java.lang.Integer").getMethod("valueOf",Object.clas

System.out.println(x); //25
```

- چون تایپ آرگومان های متدها و سازنده‌ها محدود است (`int,double,String`)، برای گرفتن کلاس مربوطه می‌توانید از `switch-case` یا `if-else` استفاده کنید.

کاربرد نمونه

مثلا با اجرای کد زیر خروجی های کامنت شده مدنظر خواهد بود:

```

public static void main(String[] args) throws Exception {

        String x = (String) RemoteCaller.creator("create java.lang.String Str
        System.out.println(RemoteCaller.caller("call replaceAll String e, Str


        StringBuilder sb = (StringBuilder) RemoteCaller.creator("create java.
        RemoteCaller.caller("call reverse",sb);

        System.out.println(RemoteCaller.caller("call toString", sb)); // nam-
        System.out.println(RemoteCaller.caller("call charAt int 1",sb)); // a


}
```

برای راهنمایی بیشتر لطفا تست نمونه را از این لینک دانلود، بررسی و اجرا کنید.