Domáca úloha č. 2

(Termín odovzdania úlohy: **do pondelka 27. marca 2023, 9:50**, t. j. do začiatku siedmej prednášky.)

Rozhranie `Iterator<E>` nie je implementované iba pomocnými triedami pre iterátory cez zoskupenia objektov – príkladom triedy implementujúcej `Iterator<String>` je napríklad aj `Scanner`. Cieľom tejto úlohy bude napísať niekoľko tried reprezentujúcich generátory (vo všeobecnosti nekonečných) postupností, ktoré zároveň budú implementáciami rozhrania `Iterator<E>`. Postupnosti budú vždy indexované prirodzenými číslami, a to od nuly.

Vytvorte balík `sequences` obsahujúci nasledujúce triedy a rozhrania (požiadavky na tieto triedy a rozhrania sú uvedené nižšie):

- Generické rozhranie `SequenceGenerator<E>` rozširujúce rozhranie `Iterator<E>` a reprezentujúce generátor nekonečnej postupnosti prvkov typu `E`.
- Generickú triedu `Recurrence<E>` implementujúcu rozhranie `SequenceGenerator<E>` a reprezentujúcu generátor postupnosti prvkov typu E danej počiatočnými hodnotami $a_0, \ldots, a_{k-1}$ pre nejaké nenulové prirodzené číslo k a rekurentným vzťahom

  $$a_{n+k} = F(a_{n+k-1}, a_{n+k-2}, \ldots, a_n)$$

  pre všetky prirodzené čísla n, kde F je nejaké zobrazenie.
- Triedu `FibonacciNumbers` rozširujúcu triedu `Recurrence<BigInteger>` a reprezentujúcu generátor postupnosti *Fibonacciho čísel* danej počiatočnými hodnotami $F_0 = 0$, $F_1 = 1$ a rekurentným vzťahom

  $$F_{n+2} = F_{n+1} + F_n$$

  pre všetky prirodzené čísla n.
- Triedu `LucasNumbers` rozširujúcu triedu `Recurrence<BigInteger>` a reprezentujúcu generátor postupnosti *Lucasových čísel* danej počiatočnými hodnotami $L_0 = 2$, $L_1 = 1$ a rekurentným vzťahom

  $$L_{n+2} = L_{n+1} + L_n$$

  pre všetky prirodzené čísla n.

Rozhranie `SequenceGenerator<E>`

Toto generické rozhranie by malo rozširovať rozhranie `Iterator<E>`. Všetky triedy implementujúce rozhranie `SequenceGenerator<E>` teda budú musieť obsahovať implementáciu metód `hasNext` a `next` deklarovaných v rozhraní `Iterator<E>`. Pre ľubovoľnú takúto triedu bude metóda `next` vracať – pokiaľ nebola volaná metóda `setNextIndex` opísaná nižšie – vždy nasledujúci ešte nevygenerovaný prvok postupnosti, ktorej generátor daná trieda reprezentuje. Prvé volanie metódy `next` teda vráti nultý prvok postupnosti, druhé volanie vráti prvý prvok postupnosti, atď. V prípade, že žiaden ďalší prvok postupnosti neexistuje, bude metóda `next` vyhadzovať výnimku typu `NoSuchElementException`. Metóda `hasNext` bude vracať `true` práve vtedy, keď nejaký ďalší prvok postupnosti existuje, t. j. keď metóda `next` pri svojom nasledujúcom volaní nevyhodí výnimku. Zvyšné metódy deklarované v rozhraní `Iterator<E>` nebude potrebné implementovať.
Okrem toho bude rozhranie `SequenceGenerator<E>` deklarovať ďalšie dve metódy:

- Metódu `void setNextIndex(int index)`, ktorá bude v triedach implementujúcich generické rozhranie `SequenceGenerator<E>` realizovať posun iterátora *pred prvok* postupnosti s indexom `index`. To znamená, že nasledujúce volanie metódy `next` vráti prvok postupnosti s indexom `index` (alebo vyhodí výnimku v prípade, že takýto prvok neexistuje).

  Implementácie tejto metódy by mali vyhadzovať výnimku typu `IllegalArgumentException` v prípade, že bude ako argument zadané záporné číslo. V opačnom prípade by táto metóda nikdy výnimku vyhadzovať nemala.

- Metódu `E getValue(int index)`, ktorá bude v triedach implementujúcich `SequenceGenerator<E>` vracať na výstupe prvok postupnosti s indexom `index` bez toho, aby sa posunul iterátor – prípadné ďalšie volania metód `next` a `hasNext` teda vyústia v rovnaké výstupy bez ohľadu na to, či resp. koľkokrát sa volala metóda `getValue`.

  Implementácie tejto metódy by mali vyhadzovať výnimku typu `IllegalArgumentException` v prípade, že je ako argument `index` zadané záporné číslo a výnimku typu `NoSuchElementException` v prípade, že je argument `index` síce nezáporný, avšak prvok postupnosti s indexom `index` neexistuje.

Trieda `Recurrence<E>`

Táto trieda implementujúca rozhranie `SequenceGenerator<E>` by mala poskytovať verejný konštruktor s dvoma parametrami. Prvým parametrom bude inštancia `initialValues` typu `List<E>` reprezentujúca počiatočné hodnoty generovanej postupnosti a druhým inštancia `formula` typu `Function<List<E>, E>` reprezentujúca zobrazenie F, pomocou ktorého sa budú počítať nasledujúce prvky postupnosti.[1]
V prípade, že je niektorý z argumentov konštruktora rovný `null`, rovnako ako v prípade nulovej dĺžky zoznamu `initialValues`, vyhodí tento konštruktor výnimku typu `IllegalArgumentException`. V opačnom prípade sa korektne inicializuje generátor rekurentnej postupnosti prvkov typu `E` takej, že:

- Počiatočné prvky $a_0, \ldots, a_{k-1}$ tejto postupnosti sú dané prvkami zoznamu `initialValues` s rovnakými indexmi (číslo k je teda dané dĺžkou zoznamu `initialValues`).
- Pre všetky $n \in \mathbb{N}$ sa prvok a_{n+k} tejto rekurentnej postupnosti získa volaním metódy `apply` funkcie `formula` na vstupnom zozname pozostávajúcom práve z prvkov $a_n, a_{n+1} \ldots, a_{n+k-1}$ (v tomto poradí).

Referenciu `null` pritom považujeme za *neexistujúci prvok postupnosti*, ktorý túto postupnosť ukončuje. Kedykoľvek teda napríklad metódou `apply` funkcie `formula` dostaneme namiesto hodnoty a_{n+k} referenciu `null`, prvok generovanej postupnosti s indexom $n + k$ neexistuje a neexistuje ani žiaden prvok postupnosti s vyšším indexom. Podľa toho sa teda musia správať všetky metódy tohto generátora postupnosti.
Prípadná neskoršia zmena v zozname `initialValues` z argumentu konštruktora triedy `Recurrence<E>` by nijako nemala ovplyvniť vnútorný stav vytvorenej inštancie tejto triedy.
Trieda ďalej musí implementovať všetky štyri metódy vyžadované rozhraním `SequenceGenerator<E>`. Tieto metódy by sa mali správať podľa špecifikácie uvedenej vyššie. V metóde `next` negenerujte zakaždým odznova všetky prvky postupnosti s nižším indexom, ale pokračujte v generovaní tam, kde ste pri minulom volaní tejto metódy skončili. V rámci inštancie triedy `Recurrence<E>` si napríklad môžete pamätať k prvkov postupnosti, ktoré sa majú vygenerovať pri nasledujúcich volaniach metódy `next` a pri každom volaní metódy `next` môžete tieto prvky aktualizovať.
Metódu `setNextIndex` môžete implementovať ako obnovenie „východzieho stavu“ generátora nasledované vhodným počtom volaní metódy `next`.[2] Metódu `getValue` môžete implementovať pomocou vytvorenia novej nezávislej inštancie triedy `Recurrence<E>` a volania metódy `setNextIndex` pre túto inštanciu.

Triedy `FibonacciNumbers` a `LucasNumbers`

Obidve tieto triedy by mali rozširovať triedu `Recurrence<BigInteger>`, pričom by mali zodpovedať postupnostiam prvkov typu `BigInteger` opísaným vyššie. Inštanciu každej z týchto tried by malo byť možné vytvoriť volaním konštruktora bez parametrov.
V rámci implementácie týchto tried zbytočne neopakujte kód z triedy `Recurrence<E>` – naopak v čo možno najväčšej miere využite dedenie. V skutočnosti by malo byť postačujúce implementovať iba konštruktor týchto dvoch tried.

---

[1] Generické rozhranie `Function<T,R>` z balíka `java.util.function` deklaruje jedinú abstraktnú metódu `apply`, ktorá na základe vstupnej hodnoty typu `T` vypočíta výstupnú hodnotu typu `R`. Každú inštanciu triedy implementujúcej rozhranie `Function<T,R>` tak možno chápať ako funkciu z `T` do `R` danú metódou `apply` tejto triedy. (Typicky môže ísť o inštancie anonymných tried alebo o tzv. lambda výrazy, s ktorými budeme pracovať neskôr počas semestra.)
Inštancia `formula` typu `Function<List<E>, E>` teda v tomto zmysle zodpovedá funkcii, ktorá na základe zoznamu hodnôt typu `E` vypočíta jedinú hodnotu typu `E`. Prvky vstupného zoznamu pritom postupne zodpovedajú prvkom $a_n, a_{n+1}, \ldots, a_{n+k-1}$ generovanej postupnosti a výstupná hodnota zodpovedá prvku a_{n+k}.

[2] Treba si však dať pozor na ošetrenie prípadných výnimiek, ktoré môže vyhadzovať metóda `next`, ale nemala by ich vyhadzovať metóda `setNextIndex`.

Odovzdávanie na testovač a kritériá hodnotenia

Súčasťou hodnotenia úlohy budú okrem správnosti riešenia aj nasledujúce aspekty:

- Zmysluplnosť objektového návrhu. Nemalo by napríklad dochádzať k opakovaniu rozsiahlejších častí rovnakého kódu v rôznych triedach alebo metódach. Tiež by mal byť dodržaný princíp zapuzdrenia.
- Rešpektovanie konvencií jazyka Java.
- Používanie anotácie `@Override` pri prekrývaní metód.
- Programátorsky štýl a čitateľnosť odovzdaného kódu.

Na testovač odovzdávajte ZIP archív obsahujúci priečinok `sequences` a v ňom všetky vaše triedy ako súčasť balíka `sequences`. Všetky bežné (to jest iné ako vnorené, lokálne, alebo anonymné) triedy a rozhrania by mali byť uložené v samostatnom súbore.