Cvičenia č. 3, úloha č. 5

Úlohou bude naprogramovať niekoľko robotov pohybujúcich sa na obdĺžnikovej mriežke, v ktorej sú políčka v nultom riadku alebo stĺpci považované za susedné s políčkami v poslednom riadku resp. stĺpci – po posune z políčka v poslednom stĺpci smerom doprava sa tak napríklad robot ocitne v nultom stĺpci. Túto situáciu si možno predstaviť tak, že sa namiesto v rovine robot pohybuje po povrchu toru.[1] Na každom políčku takéhoto toru je uložená booleovská hodnota `true` alebo `false`. Robot v každom svojom kroku dostane hodnotu uloženú na políčku, na ktorom sa momentálne nachádza – na základe tejto informácie potom môže túto hodnotu prepísať a pohnúť sa ľubovoľným zo štyroch prípustných smerov (doľava, nahor, doprava, alebo nadol), prípadne môže aj zostať na mieste.

Najdôležitejšie súčasti priloženého archívu

Stiahnite si priložený archív obsahujúci balík `robots` a v ňom niekoľko tried, spomedzi ktorých sú niektoré hotové a iné určené na doimplementovanie. Všetky triedy reprezentujúce robotov musia implementovať už hotové rozhranie `Robot` s nasledujúcimi dvoma metódami:

- Metódou `Move nextMove(Boolean currentCellContents)`, ktorá na základe informácie o booleovskej hodnote uloženej na políčku, na ktorom sa robot v danom momente nachádza, vráti nasledujúci krok robota v podobe inštancie už hotovej triedy `Move` opísanej nižšie.

  V prípade, že už robot v danej situácii nemá definovaný žiaden ďalší krok, malo by dôjsť k vyhodeniu výnimky typu `NoNextMoveException` – táto podtrieda triedy `RuntimeException` je už taktiež hotová.

- Metódou `boolean hasNextMove(Boolean currentCellContents)`, ktorá vráti `true` práve vtedy, keď daný robot v prípade, že stojí na políčku s booleovskou hodnotou `currentCellContents`, môže pokračovať ďalším krokom – t. j. presne v prípade, keď ďalšie volanie metódy `nextMove` s argumentom `currentCellContents` nepovedie k vyhodeniu výnimky.

Hotová trieda `Move`, ktorej inštancie reprezentujú pohyby robotov, je v princípe „obalom“ pre dve nemodifikovateľné položky, ku ktorým možno pristupovať pomocou ich metód `get`:

- Novú booleovskú hodnotu `newCellContents`, ktorú robot v danom kroku na dané políčko zapíše.
- Smer pohybu robota `direction` – ide tu o inštanciu vymenovaného typu `Direction`, ktorý môže nadobúdať hodnoty `LEFT`, `UP`, `RIGHT`, `DOWN` a `STAY`, postupne zodpovedajúce pohybom doľava, nahor, doprava a nadol, resp. žiadnemu pohybu.[2]

V triede `Move` sú prekryté aj metódy `equals` a `hashCode` z triedy `Object` – význam týchto metód si vysvetlíme na piatej prednáške a pri riešení tejto úlohy ich môžete ignorovať.

V balíku `robots` je už tiež hotová generická trieda `StateSetRobot<T>` implementujúca rozhranie `Robot`. Inštancie tejto triedy reprezentujú robotov, pohyby ktorých okrem obsahov jednotlivých políčok toru závisia už len na ich *stave*, ktorým môže byť ľubovoľná inštancia typu daného typovým parametrom `T`. Napríklad robot typu `StateSetRobot<Integer>` má celočíselné stavy, atď.
Pohyby takéhoto robota sú plne určené jeho *programom* – ním rozumieme inštanciu hotového generického rozhrania `StateSetRobotProgram<T>`, ktorú robot typu `StateSetRobot<T>` dostane ako jediný argument svojho konštruktora. Toto rozhranie deklaruje dve metódy:

- Metódu `T getInitialState()`, ktorá vráti *počiatočný stav* robota s daným programom. Ak je teda robot typu `StateSetRobot<T>` vytvorený pomocou volania konštruktora s argumentom `program`, je na začiatku simulácie robot vždy v stave `program.getInitialState()`.

[1] *Torus* je geometrická plocha podobná šiške alebo pneumatike: `https://en.wikipedia.org/wiki/Torus`.

[2] O vymenovaných typoch sa možno dočítať napríklad tu: `https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html`. Pre účely tejto úlohy je podstatné iba toľko, že k hodnotám vymenovaného typu `Direction` možno pristupovať pomocou syntaxe `Direction.LEFT`, `Direction.UP`, atď., pričom pre vymenované typy funguje aj `switch` (príklad možno nájsť v metóde `start` hotovej triedy `Simulation`).

- Metódu

  ```
  StateSetRobotStep<T> nextStep(T currentState, Boolean currentCellContents)
  ```

  – tú možno chápať ako realizáciu zobrazenia, ktoré pre daný stav robota `currentState` typu `T` a obsah čítaného políčka `currentCellContents` typu `Boolean` vráti nasledujúcu akciu robota. Táto nasledujúca akcia je pritom reprezentovaná ako inštancia hotovej generickej triedy `StateSetRobotStep<T>`, slúžiacej ako „obal“ pre dve nemodifikovateľné položky prístupné pomocou ich metód `get`:

  - Inštanciu `move` vyššie opísanej triedy `Move`, udávajúcej pohyb robota v príslušnom kroku (čiže hodnotu zapísanú na pôvodne čítané políčko a smer, ktorým sa robot v tomto kroku vydá).
  - Nový stav robota `newState` typu `T`.

  Metódy `equals` a `hashCode` v triede `StateSetRobotStep<T>` možno opäť ignorovať.

  V prípade, že pre nejaký stav a obsah políčka nie je programom definovaná žiadna nasledujúca akcia robota, mala by metóda `nextStep` vrátiť referenciu `null` (metóda `nextMove` robota typu `StateSetRobot<T>` v takom prípade vyhodí výnimku).

V balíku `robots` je tiež hotová trieda `Run` s metódou `main` realizujúcou načítanie vstupu v podobe detailne opísanej nižšie – vstupom sú rozmery a počiatočné obsahy jednotlivých políčok toru, typ vytváraného robota a jeho prípadná ďalšia špecifikácia, ako aj počiatočná pozícia robota na tore. Na základe týchto údajov sa vytvorí inštancia triedy `Simulation` realizujúcej samotnú simuláciu pohybov robota na tore, ktorá sa následne spustí jej metódou `start`.

Vlastné zadanie

Implementujte nasledujúce triedy reprezentujúce programy pre robotov typu `StateSetRobot`, ktorých kostry sa už v balíku `robots` nachádzajú:

- Generickú triedu `FiniteStateSetRobotProgram<T>` implementujúcu pre jej typový parameter `T` rozhranie `StateSetRobotProgram<T>`. Tá bude reprezentovať *konečné* programy, v ktorých je pre niekoľko pevne daných dvojíc pozostávajúcich zo stavu robota a obsahu čítaného políčka priamo definovaný výstup v podobe inštancie vyššie opísanej triedy `StateSetRobotStep<T>`. Vstupné dvojice sú pritom reprezentované ako inštancie hotovej generickej triedy `StateSetRobotConfiguration<T>` slúžiacej ako „obal“ pre nasledujúce dve nemodifikovateľné položky prístupné pomocou ich metód `get`:

  - Stav robota `currentState` typu `T`.
  - Booleovský obsah `currentCellContents` políčka, na ktorom sa robot nachádza.

  Metódy `equals` a `hashCode` možno aj v tejto triede ignorovať.

  Trieda `FiniteStateSetRobotProgram<T>` by mala poskytovať konštruktor umožňujúci vytvoriť takýto program na základe zobrazenia `transitions` typu

  ```
  Map<StateSetRobotConfiguration<T>, StateSetRobotStep<T>>
  ```

  a hodnoty `initialState` typu `T`. Výsledný program by mal mať počiatočný stav `initialState` a výstupy metódy `nextStep` by mali byť dané zobrazením `transitions`: pre každý vstupný stav `currentState` a booleovskú hodnotu `currentCellContents` sa výstup metódy `nextStep` získa „zabalením“ týchto dvoch vstupov do inštancie triedy `StateSetRobotConfiguration<T>` a následným použitím hodnoty prislúchajúcej k tejto inštancii v zobrazení `transitions`. V prípade, že daná inštancia triedy `StateSetRobotConfiguration<T>` nie je kľúčom v zobrazení `transitions`, má byť výstupom metódy `nextStep` referencia `null`.

- Triedu `CellInvertingProgram` implementujúcu rozhranie `StateSetRobotProgram<Integer>`. Jej inštancie by mali reprezentovať programy robotov vykonávajúcich konečne veľa krokov, v ktorých sa zakaždým udeje nasledujúce:
  - Obsah príslušného políčka sa zneguje – t. j. `true` sa zmení na `false` a naopak.
  - V prípade, že bola *pôvodne* na políčku zapísaná hodnota `true`, pohne sa robot nadol; v opačnom prípade sa pohne doprava.

  Jediným argumentom konštruktora tejto triedy je celé číslo `totalSteps` udávajúce počet krokov vyššie uvedeného typu, ktoré bude robot vykonávať. To znamená, že prvých `totalSteps` volaní metódy `nextMove` robota s týmto programom by malo mať za následok vyššie opísané správanie a vo zvyšných volaniach by malo dôjsť k vyhodeniu výnimky (spôsobenej tým, že metóda `nextStep` programu vráti referenciu `null`).

  Stavy robota možno zvoliť ľubovoľne – jediným obmedzením je, že musí ísť o celé čísla.

- Triedu `MarkInitialCellProgram` implementujúcu rozhranie `StateSetRobotProgram<String>`. Jej inštancie budú reprezentovať jediný pevne daný program robota, ktorý prepíše obsahy jednotlivých políčok toru tak, aby po jeho zastavení:
  - Bola na počiatočnom políčku, z ktorého robot vychádzal, zapísaná hodnota `true`.
  - Na všetkých ostatných políčkach toru bola zapísaná hodnota `false`.

  Po dokončení tejto úlohy sa robot s týmto programom zastaví – čo znamená, že jeho metóda `nextMove` bude vyhadzovať výnimku (spôsobenú tým, že metóda `nextStep` programu vráti referenciu `null`).

  Inštanciu tejto triedy by malo byť možné vytvoriť pomocou konštruktora bez parametrov (ten nemusí byť definovaný explicitne – postačí aj automaticky vygenerovaný konštruktor).

  Treba počítať s tým, že počiatočné políčko, ako aj počiatočné obsahy políčok toru, môžu byť ľubovoľné. Na presnej postupnosti krokov, pomocou ktorej sa uvedená transformácia vykoná, pri tejto triede nezáleží. Stavy robota možno tiež zvoliť ako ľubovoľné reťazce typu `String`.

Ďalej implementujte generickú triedu `ReversedStateSetRobot<T>` rozširujúcu triedu `StateSetRobot<T>`. Podobne ako pri robotoch typu `StateSetRobot<T>`, malo by byť možné aj inštanciu tejto triedy vytvoriť pomocou konštruktora s jediným argumentom `program` typu `StateSetRobotProgram<T>` reprezentujúcim program vytvoreného robota. Metóda `nextMove` ale bude vykonávať tento program tak, že všetky pohyby nahradí pohybmi opačným smerom – t. j. namiesto pohybov doľava bude realizovať pohyby doprava, namiesto pohybov nahor pohyby nadol, namiesto pohybov doprava pohyby doľava a namiesto pohybov nadol pohyby nahor. Metóda `hasNextMove` by mala byť s touto metódou konzistentná.
V triede `ReversedStateSetRobot<T>` *neimplementujte* celú triedu `StateSetRobot<T>` odznova – naopak sa snažte v maximálnej miere využiť dedenie. Špeciálne by malo byť možné napísať túto triedu aj bez toho, aby sa v premennej jej inštancie uchovával vykonávaný program.

Formát vstupu a výstupu

Načítanie vstupu realizuje hotová metóda `main` triedy `Run` a metódy, ktoré sú z metódy `main` volané. Formát vstupu je nasledujúci:

- Vstup sa začína riadkom s dvojicou prirodzených čísel `height` a `width` oddelených medzerou. Tie reprezentujú počet riadkov resp. počet stĺpcov toru, na ktorom bude simulácia prebiehať.

  Môžete predpokladať, že `height` aj `width` budú vždy kladné prirodzené čísla.

- Nasleduje presne `height` riadkov, pričom každý z nich pozostáva z reťazca dĺžky `width` zloženého zo znakov „`T`“ a „`F`“ zodpovedajúcich booleovským hodnotám `true` resp. `false`. Tieto riadky obvyklým spôsobom udávajú hodnoty uložené na jednotlivých políčkach toru na začiatku simulácie.

- Nasledujú dva riadky určujúce typ vytváraného robota. Prvý z nich obsahuje jeden z reťazcov
  - `"FiniteStateSetRobotProgram-Integer"`,
  - `"FiniteStateSetRobotProgram-String"`,
  - `"CellInvertingProgram"`, alebo
  - `"MarkInitialCellProgram"`;

  tie určujú typ programu použitého na vytvorenie robota (v prvých dvoch prípadoch vrátane typového parametra `Integer` alebo `String`). Na ďalšom riadku potom nasleduje reťazec `"normal"` alebo `"reversed"` určujúci typ samotného robota – v prvom prípade sa vytvorí robot typu `StateSetRobot<Integer>` resp. `StateSetRobot<String>` (podľa typu použitého programu) a v druhom prípade sa vytvorí robot `ReversedStateSetRobot<Integer>` alebo `ReversedStateSetRobot<String>`.

  Ak je na prvom z týchto riadkov `"MarkInitialCellProgram"`, bude na druhom z nich vždy `"normal"`.

- Môžu nasledovať riadky bližšie udávajúce parametre vytvoreného robota:
  - V prípade použitia programu `MarkInitialCellProgram` táto časť vstupu chýba.
  - V prípade použitia programu `CellInvertingProgram` ide o jediný riadok obsahujúci celé číslo `totalSteps` použité ako argument konštruktora triedy `CellInvertingProgram`. Môžete predpokladať, že toto číslo je vždy nezáporné.
  - Pri programe `FiniteStateSetRobotProgram<T>`, kde `T` je `Integer` alebo `String`, sa táto časť vstupu začína prirodzeným číslom `transitionCount`, za ktorým nasleduje `transitionCount` riadkov postupne reprezentujúcich všetky dvojice kľúčov a hodnôt zobrazenia `transitions`, ktoré sa použije ako argument konštruktora programu. Každá takáto dvojica je na vstupe zadaná v podobe

    `currentState currentCellContents newState newCellContents direction`,

    kde prvé dve položky reprezentujú vstupný stav a obsah políčka toru a zvyšné položky reprezentujú výstup v podobe nového stavu, nového obsahu políčka a smeru pohybu robota. Za týmito riadkami nasleduje ešte jeden riadok, na ktorom je uvedený počiatočný stav typu `T`.

- Na ďalšom riadku nasledujú dve prirodzené čísla – riadok a stĺpec – reprezentujúce počiatočnú pozíciu vytváraného robota na tore.

- Zvyšok vstupu určuje parametre spúšťanej simulácie: na predposlednom riadku vstupu je počet simulovaných krokov robota (ak je záporný, simuluje sa až dovtedy, kým sa robot zastaví) a na poslednom riadku je jeden z reťazcov `"steps"` alebo `"cells"` určujúcich formát výstupu. V prvom prípade bude na výstup vypísaná celá postupnosť krokov simulovaného robota; v zostávajúcom prípade sa po ukončení simulácie na výstup vypíše obsah všetkých políčok toru v podobnom formáte, v akom bol ich počiatočný obsah načítaný zo vstupu. Pri programoch `FiniteStateSetRobotProgram<T>` a `CellInvertingProgram` sa na testovači vždy použije formát `"steps"` (keďže takéto roboty majú zadaním presne predpísanú postupnosť krokov) a pri programe `MarkInitialCellProgram` sa použije formát `"cells"` (keďže zadanie určuje iba výslednú podobu toru).

Odovzdávanie na testovač

Na testovač odovzdávajte ZIP archív obsahujúci priečinok `robots` a v ňom všetky triedy balíka `robots` (vrátane tých, ktoré už boli hotové). Všetky triedy by mali byť uložené v samostatnom súbore. Testovač bude kontrolovať správnosť vášho riešenia na vybraných vstupoch.

Príklad vstupu č. 1:

```
7 7
FFFTFFF
TTTFTTT
FFFTFFF
TTTFTTT
FFFTFFF
TTTFTTT
FFFTFFF
FiniteStateSetRobotProgram-Integer
normal
8
0 false 0 true RIGHT
0 true 1 true DOWN
1 false 1 false LEFT
1 true 2 true UP
2 true 2 true LEFT
2 false 3 true STAY
3 false 3 false STAY
3 true 4 false STAY
0
0 0
-1
steps
```

Príklad výstupu č. 1:

```
true RIGHT
true RIGHT
true RIGHT
true DOWN
false LEFT
true UP
true LEFT
true LEFT
true LEFT
true STAY
false STAY
```

Príklad vstupu č. 2:

```
4 4
FFFF
FFFF
FFFF
FFFF
CellInvertingProgram
normal
10
1 2
-1
steps
```

Príklad výstupu č. 2:

```
true RIGHT
true RIGHT
true RIGHT
true RIGHT
false DOWN
true RIGHT
true RIGHT
true RIGHT
true RIGHT
false DOWN
```

Príklad vstupu č. 3:

```
4 4
FFFF
FFFF
FFFF
FFFF
CellInvertingProgram
reversed
10
1 2
-1
steps
```

Príklad výstupu č. 3:

```
true LEFT
true LEFT
true LEFT
true LEFT
false UP
true LEFT
true LEFT
true LEFT
true LEFT
false UP
```

Poznámka: skutočnosť, že ide o postupnosť krokov z predchádzajúceho príkladu s „obrátenými“ smermi pohybu, je iba zhodou okolností – vo všeobecnosti to tak byť nemusí.

Príklad vstupu č. 4:

```
5 7
TFTFTFT
TFTFTFT
TFTFTFT
TFTFTFT
TFTFTFT
MarkInitialCellProgram
normal
1 2
-1
cells
```

Príklad výstupu č. 4:

```
FFFFFFF
FFTFFFF
FFFFFFF
FFFFFFF
FFFFFFF
```