ជំពូកទី ១០ : អំពី Collections

១. អំពី Collection Classes

ពេលនេះគេគួរតែស្គាល់ collection interfaces ហើយ និងត្រៀមលក្ខណៈជាស្រេច ដើម្បីពិនិត្យមើល classes ធម្មតា និងការប្រើ collection interfaces។ classes មួយចំនួន ផ្តល់នូវការអនុវត្តពេញលេញដែលអាចប្រើបានតែម្តង។ classes មួយចំនួនទៀតមាន

លក្ខណៈជា abstract class ផ្តល់នូវការអនុវត្តជាគំរោង ដែលត្រូវបានប្រើជាចំណុចចាប់ផ្តើមសំរាប់បង្កើតពពួក collections ដែលគ្មានលក្ខណៈជា abstract class នោះ ។ វាពុំមាន collection classes ដែលមានលក្ខណៈជា synchronized ឡើយ ប៉ុន្តែយើងនឹងឃើញនៅពេលក្រោយ គឺវាអាចមានការប្រើជាលក្ខណៈ synchronized ។

collection classes ធម្មតាបានរាយឈ្មោះដូចខាងក្រោម:

ឈ្មោះ class	ការអធិប្បាយ
AbstractCollection	ភាគច្រើនប្រើ Collection interface ។
AbstractList	ទទួលលក្ខណៈពី AbstractCollection ហើយភាគច្រើនប្រើ List interface ។
AbstractSequentialList	ទទួលលក្ខណៈពី AbstractList សំរាប់ប្រើជាមួយ collection ដែលប្រើលក្ខណៈបន្តគ្នា គ្រាន់ជាងប្រើលក្ខណៈមិនតាមលំដាប់លំដោយនៃធាតុរបស់វា ។

LinkedList	ប្រើ linked list ដោយទទួលលក្ខណៈពី AbstractSequentialList ។
ArrayList	ប្រើ dynamic array ដោយទទួលលក្ខណៈពី AbstractList ។
AbstractSet	ទទួលលក្ខណៈ AbstractCollection និងភាគច្រើនប្រើ Set interface ។
HashSet	ទទួលលក្ខណៈពី AbstractSet សំរាប់ប្រើ hash table ។
TreeSet	ទទួលលក្ខណៈពី AbstractSet ដែលប្រើ set ផ្ទុកក្នុង tree មួយ ។

២. អំពី ArrayList class

ArrayList class ទទួលលក្ខណៈពី AbstractList ហើយប្រើ List interface ។ ArrayList ផ្តល់នូវ dynamic arrays ដែលអាចពង្រីកតាមតំរូវការ ។ ក្នុង Java, array ធម្មតាគឺមានប្រវែងថេរ ។ បន្ទាប់ពី arrays ត្រូវបានបង្កើតឡើងនោះ គេមិនអាចពង្រីកឬបង្រួមហើយគេត្រូវដឹងជាមុននូវចំនួនធាតុ array នឹងត្រូវផ្ទុក ។ ក៏ប៉ុន្តែជួនកាលគេអាចមិនដឹងរហូតដល់ពេលដំណើរការចំពោះទំហំ array ដែលគេត្រូវការ ។

ដើម្បីប្រើសភាពបែបនេះ គំរោងការ collections បានកំណត់នូវ ArrayList ។ ជាសំខាន់ ArrayList គឺជាអញ្ញាត array មួយដែលមានប្រវែងនៃ object references ។ បានន័យថា ArrayList អាចបង្កើនឬបន្ថយទំហំតាមតំរូវការ ។ Array list ត្រូវបានបង្កើតឡើងដោយមានទំហំដំបូងមួយ ។ នៅពេលគេប្រើទំហំនេះហួស collection នឹងពង្រីកបន្ថែមដោយស្វ័យប្រវត្ត ។ នៅពេល objects ត្រូវបានលុបបំបាត់ នោះ array អាចបង្រួមមកវិញ ។

ArrayList មាន constructors ដូចបានបង្ហាញខាងក្រោមនេះ :

ArrayList()
ArrayList(Collection *c*)
ArrayList(int *capacity*)

ទំរង់ constructor ទីមួយបង្កើត array list ទទេ មួយ ។ ទំរង់ constructor ទីពីរបង្កើត array list មួយដែលបានកំណត់តំលៃដំបូងនូវធាតុនៃ collection គឺ *c* ។ ទំរង់ constructor ទីបីបង្កើតនូវ array list មួយដែលបានកំណត់តំលៃ *capacity* ដំបូង ។ capacity គឺជាទំហំបញ្ជាក់អោយ

array ដែលត្រូវបានប្រើសំរាប់ផ្ទុកធាតុ។ capacity នេះពង្រីកបន្ថែមដោយស្វ័យប្រវត្តពេលដែលធាតុបានបន្ថែមទៅលើ array list ។

កម្មវិធីខាងក្រោមនេះបង្ហាញនូវការប្រើ ArrayList យ៉ាងសាមញ្ញ។ Array list ត្រូវបានបង្កើតឡើង ពេលនោះ objects នៃប្រភេទ String ត្រូវបានបញ្ចូលទៅអោយវា។ រួចហើយ list ត្រូវបានបង្ហាញ។ ធាតុខ្លះត្រូវលុបបំបាត់ និង list ត្រូវបង្ហាញម្តងទៀត ។

គួរកត់សំគាល់ថា al ចាប់ផ្តើមដោយទទេ ហើយពង្រីកនៅពេលធាតុត្រូវបានបញ្ចូលទៅអោយវា។ នៅពេលធាតុត្រូវបានលុបបំបាត់នោះទំហំរបស់វាត្រូវថយចុះ។

នៅក្នុងឧទាហរណ៍មុននេះ តំណែលរបស់ collection មួយត្រូវបានបង្ហាញដោយប្រើការបំលែងធម្មតា ដែលបានផ្តល់តាមរយៈ toString() និងបានទទួលលក្ខណៈពី AbstractCollection។ ទោះបីជាវាមានលក្ខណៈគ្រាប់គ្រាន់ ចំពោះកម្មវិធីតំរូដ៏ខ្លីនេះ គេកំរប្រើ method នេះសំរាប់បង្ហាញ តំណែលរបស់ collection ដែលប្រើប្រាស់ក្នុងជីវភាពរស់នៅ។

ជាធម្មតា គេផ្តល់នូវដំណើរទំរង់លទ្ធផលរបស់ខ្លួន ។ ចំពោះឧទាហរណ៍នៅបន្តបន្ទាប់នេះ លទ្ធផលធម្មតាបានបង្កើតឡើងដោយ toString() ដែលនឹងត្រូវបានប្រើបន្តទៀត ។

ទោះបីជាចំណុះរបស់ ArrayList object កើនឡើងដោយស្វ័យប្រវត្តតាមដែល objects មានក្នុងវាក៏ដោយ គេអាចបង្កើនចំណុះ ArrayList object តាមលក្ខណៈធម្មតាដោយ ហៅ ensureCapacity() method មកប្រើ ។ គេអាចធ្វើបែបនេះនៅពេលគេដឹងជាមុននូវអ្វីដែលគេត្រូវផ្ទុកនូវចំនួន items ក្នុង collection ដែលកំពុងមាន ។ តាមរយៈការបង្កើននូវចំណុះ

ម្តងៗនៅត្រង់ចំណុចចាប់ផ្តើម គេអាចរក្សាទុកនូវទីតាំងជាច្រើនដែលប្រើជាថ្មីសំរាប់ពេលក្រោយ ។ ទំរង់របស់ ensureCapacity() គឺ:

void ensureCapacity(int *cap*)

ក្នុងនោះ *cap* គឺជាចំណុះថ្មី ។

បើសិនជាគេចង់បន្ថយទំហំរបស់ array ដែលក្នុងនេះគេសំដៅទៅលើ ArrayList object ដូច្នេះវាអាចពង្រីកទំហំធំតាមចំនួន items ដែលវាកំពុងមាន ដោយហៅ trimToSize() មកប្រើ ។ ទំរង់របស់វា គឺ: void trimToSize()

៣. ដើម្បីទទួលបាន array មួយពី ArrayList

នៅពេលធ្វើការជាមួយនឹង ArrayList ជួនកាលគេចង់បាន array ពិតប្រាកដមួយដែលផ្ទុកតំលៃរបស់ list ។ ដូចបានពន្យល់ខាងដើមរួចហើយ គេអាចធ្វើដូច្នេះបានដោយប្រើ toArray() ។ ហេតុផលមួយចំនួនដែលគេចង់បំលែង collection មួយទៅជា array មួយគឺ :

- ដើម្បីទទួលបានរយៈពេលដំណើរការលឿនចំពោះការប្រតិបត្តិពិតប្រាកដ ។
- ដើម្បីបញ្ជូន array មួយទៅអោយ method ដែលមិន

បានធ្វើ overloaded សំរាប់ទទួលយកនូវ collection មួយ ។ ដើម្បីបញ្ជូលនូវ code ដែលមានមូលដ្ឋានលើ collection ហើយមានសភាពថ្មីទៅលើ code ចាស់ ដែលវាមិនទាន់ស្គាល់នូវ collections ។

កម្មវិធីខាងក្រោមនេះបង្ហាញនូវការបំលែង ArrayList ទៅជា array មួយមានដូចតទៅ :

កម្មវិធីចាប់ផ្តើមដោយបង្កើតនូវ collection នៃចំនួនគត់ ។ ដូចបានពន្យល់ហើយ គេមិនអាចផ្ទុកប្រភេទទិន្នន័យទំរង់ងាយក្នុង collection មួយបានឡើយ ហេតុនេះ objects នៃប្រភេទ Integer ត្រូវបាន បង្កើតឡើងនិងត្រូវបានផ្ទុក ។ បន្ទាប់មក toArray() ត្រូវបានហៅប្រើ ហើយវាទទួលបាននូវ array នៃ Object ។ តំលៃរបស់ array នេះត្រូវបានកំណត់បញ្ជាក់ជា Integer រួចហើយតំលៃត្រូវធ្វើប្រមាណវិធីបូក ។

៤. អំពី LinkedList class

LinkedList class ទទួលលក្ខណៈពី AbstractSequentialList ហើយអនុវត្តនូវ List interface ។ វាផ្តល់នូវ data structure ប្រភេទ linked list ។

LinkedList()
LinkedList(Collection *c*)

ទំរង់ constructor ទីមួយបង្កើត linked list ទទេមួយ ។ ទំរង់ constructor ទីពីរបង្កើតនូវ linked list ដែលបាន

កំណត់តំណែងដំបូងនូវធាតុនៃ collection *c* ។

លើសពីនេះ methods ដែលវាបានទទួលពីប្រភពនោះ

LinkedList class បានកំណត់នូវ LinkedList class

សំខាន់ៗមួយចំនួនរបស់ខ្លួនវាសំរាប់ប្រើនិងចូលប្រើ lists ។

ដើម្បីបន្ថែមធាតុទៅផ្នែកខាងដើម list គេប្រើ addFirst()

ដើម្បីបន្ថែមធាតុទៅផ្នែកខាងចុង គេប្រើ addLast() ។

ទំរង់របស់វា គឺ : void addFirst(Object *obj*)

void addLast(Object *obj*)

ក្នុងនេះ *obj* គឺជា item ដែលត្រូវបញ្ចូល ។

ដើម្បីទទួលបាននូវធាតុទីមួយ គេប្រើ getFirst() ។ ដើម្បីទាញយកធាតុចុងក្រោយ គេប្រើ getLast() ។ ទំរង់របស់វា គឺ:

Object getFirst()
Object getLast()

ដើម្បីលុបបំបាត់នូវធាតុទីមួយ គេប្រើ removeFirst() ។ ដើម្បីលុបបំបាត់ធាតុចុងក្រោយ គេប្រើ removeLast() ។ ទំរង់របស់វា គឺ:

Object removeFirst()
Object removeLast()

កម្មវិធីខាងក្រោមនេះបង្ហាញនូវ methods មួយចំនួនដែលផ្តល់ដោយ LinkedList:

ដោយសារ LinkedList អនុវត្តតាម List interface វាប្រើ add(Object) ដើម្បីបន្ថែម items ទៅផ្នែកខាងចុងនៃ list ដូចប្រើ addLast() ដែរ។ ដើម្បីបញ្ចូលនូវ items ត្រង់ទីតាំងកំនត់ គេប្រើទំរង់ add(int, Object) របស់ add() ដូចបានបង្ហាញតាមរយៈការប្រើ add(1, "A2") ក្នុងឧទាហរណ៏។

គួរកត់សំគាល់ថាធាតុទីបីក្នុង ll ត្រូវផ្លាស់ប្តូរដោយប្រើ get() និង set() ។ ដើម្បីទទួលបាននូវតំលៃកំពុងប្រើរបស់ធាតុមួយ គេប្រើ get() នៅត្រង់ index ដែលធាតុនោះកំពុងផ្ទុក ។ដើម្បីកំនត់នូវតំលៃថ្មីទៅអោយ index នោះ គេប្រើ set() ហើយនិងតំលៃ

ថ្នែរបស់វា ។

៥. ការចូលប្រើ Collection តាមរយៈ Iterator មួយ

ជារឿយៗគេចង់ធ្វើការប្រើដដែលៗឆ្លងកាត់ធាតុក្នុង collection មួយ ។ ឧទា : គេចង់បង្ហាញធាតុនិមួយៗជាដើម ។ វិធីងាយបំផុតសំរាប់ធ្វើសកម្មភាពបែបនេះគឺប្រើ *iterator* ដែលជា object អនុវត្តតាម Iterator ឬ ListIterator interface ។ Iteractor អាចអោយគេធ្វើសកម្មភាពដដែលៗ ឆ្លងកាត់ collection មួយក្នុងការទទួលយកឬលុបបំបាត់ធាតុ ។

ListIterator ទទួលលក្ខណៈពី Iterator ដែលអាចអោយឆ្លងកាត់ list តាមពីរទិសដៅ ហើយធ្វើការកែប្រែធាតុ ។ Iterator interface មាន methods ដូចខាងក្រោមនេះ :

ឈ្មោះ Method	អធិប្បាយ
boolean hasNext()	អោយតំលៃ true បើសិនវានៅមានធាតុទៀត ។ ផ្ទុយទៅវិញវាអោយតំលៃ false ។
Object next()	អោយតំលៃជាធាតុដែលនៅបន្ទាប់ ។ ប្រើ NoSuchElementException បើសិនជាវាពុំមានធាតុនៅបន្ទាប់ ។

void remove() លុបបំបាត់នូវធាតុកំពុងប្រើ ។ ប្រើ IllegalStateException បើសិនជាវាមានការព្យាយាមហៅ remove() មកប្រើដែលមិនមានធាតុនៅខាងមុខ និងហៅ next() មកប្រើ ។

ចំពោះ ListIterator ក៏មាន methods មួយចំនួនដែរ ។

methods ទាំងនោះគឺ:

ឈ្មោះ Method	អធិប្បាយ
void add(Object *obj*)	បញ្ចូល *obj* ទៅក្នុង list ខាងមុខធាតុដែលនឹងអោយតំលៃតាមរយៈការហៅ next() មកប្រើ ។
boolean hasNext()	អោយតំលៃ true កាលណាវាមានធាតុបន្ទាប់ ។ ផ្ទុយទៅវិញវាអោយតំលៃ false ។

boolean hasPrevious() អោយតំលៃ true បើសិនវា

មានធាតុនៅខាងមុខ ។ ផ្ទុយទៅវិញវាអោយ

តំលៃ false ។

Object next() អោយតំលៃជាធាតុដែលនៅបន្ទាប់ ។ ប្រើ

NoSuchElementException

បើសិនជាវាពុំមានធាតុនៅបន្ទាប់ ។

int nextIndex() អោយតំលៃជា index នៃធាតុដែល

នៅបន្ទាប់ ។ បើសិនវាពុំមានធាតុនៅបន្ទាប់

ទេនោះវាអោយទំហំនៃ list ។

Object previous() អោយតំលៃជាធាតុនៅខាងមុខ ។

ប្រើ NoSuchElementException

បើសិនជាវាពុំមានធាតុនៅខាងមុខទេ ។

int previousIndex() អោយតំលៃជា index នៃធាតុ

នៅខាងមុខ ។ បើវាពុំមានធាតុនៅខាងមុខទេ

នោះវានឹងអោយតំលៃ -1 ។

void remove() លុបបំបាត់នូវធាតុកំពុងប្រើ ។ ប្រើ IllegalStateException បើសិនជាវា មានការព្យាយាមហៅ remove() មកប្រើ មុន next() ឬ previous() ត្រូវបានប្រើ ។

void set(Object *obj*) កំណត់ *obj* ទៅអោយធាតុកំពុង ប្រើ ។ វាជាធាតុចុងក្រោយអោយតំណែ ដោយហៅ next() ឬ previous() មកប្រើ ។

៦. ការប្រើ iterator

មុននឹងគេអាចចូលប្រើ collection មួយតាមរយៈ iterator គេត្រូវតែមាន collection class មួយ។ collection class និមួយៗផ្តល់នូវ iterator() method ដែលអោយតំលៃជា iterator មួយចំពោះ ការចាប់ផ្តើមនៃ collection។ តាមរយៈការប្រើ iterator object គេអាចចូលប្រើធាតុនិមួយៗក្នុង collection ម្តងមួយធាតុ។ ជាទូទៅ ដើម្បីប្រើ iterator មួយ សំរាប់ដំណើរការដដែលៗតាមរយៈតំលៃនៃ collection ដោយអនុវត្តតាមជំហានខាងក្រោមនេះ :

១ - ទទួលបាន iterator មួយសំរាប់ចាប់ផ្តើមនៃ collection តាមរយៈនៃការហៅ iterator របស់ collection មកប្រើ ។

២ - បង្កើត loop ដែលមានការហៅប្រើ hasNext() ។ ត្រូវ មាន loop ធ្វើសកម្មភាពដដែលៗ អោយបានច្រើនដរាបណា hasNext() នៅតែអោយតំលៃ true ។

៣ - នៅក្នុង loop គេទទួលបាននូវធាតុនិមួយៗតាមរយៈការហៅ next() មកប្រើ ។

ចំពោះ collections ដែលប្រើ List គេអាចមាន iterator មួយដែលដោយហៅ ListIterator មកប្រើ។ ដូចបានពន្យល់ហើយថា list iterator ផ្តល់អោយគេនូវលទ្ធភាពចូលប្រើ collection ទាំងទិសដៅទៅមុខនិងទិសដៅមកក្រោយ ហើយអាចអោយគេកែប្រែនូវធាតុមួយបាន។ ផ្ទុយទៅវិញ ListIterator ត្រូវបានប្រើដូចនឹង Iterator ដែរ។

ឧទាហរណ៍ខាងក្រោមអនុវត្តជំហានទាំងនេះនិងបង្ហាញពីការប្រើទាំង Iterator និង ListIterator។

រ៉ាប្រើ ArrayList object ក៏ប៉ុន្តែគោលការណ៍ទូទៅប្រើប្រភេទ ទិន្នន័យណាមួយរបស់ collection ។ តាមពិត ListIterator អាចប្រើបានតែចំពោះ collections ដែលអនុវត្តតាម List interface ។

៧. អំពី Comparators

ទាំង TreeSet និង TreeMap សុទ្ធតែផ្ទុកធាតុតាមលំដាប់ ។ ទោះជាយ៉ាងណាក៏ដោយ វាជា comparator ដែលបង្កើតនូវ មធ្យោបាយ "រៀបតាមលំដាប់" យ៉ាងជាក់លាក់ ។ តាមធម្មតា classes ទាំងនេះផ្ទុកនូវធាតុរបស់វាដោយប្រើនូវអ្វីដែល Java ចង់បញ្ជាក់សំដៅទៅលើ "លំដាប់តាមក្រឹត្យក្រម" ដែលតាមធម្មតា គឺជាលំដាប់ដែលគេបានរំពឹងគិតទុក (A នៅពីមុខ B. 1 នៅពីមុខ ២ -ល-) ។

បើសិនជា គេចង់រៀបធាតុតាមលំដាប់ក្នុងវិធីផ្សេង ពេលនោះគេកំណត់នូវ object នៃ Comparator នៅពេលគេ បង្កើតឡើងនូវសំណុំ ឬ map ។ ទម្លាប់បែបនេះផ្តល់អោយគេនូវលទ្ធភាពដើម្បីដឹកនាំទៅរកវិធីផ្ទុកធាតុក្នុង collection ដែលបានរៀបនិង map ។

Comparator interface មាន methods ពីរគឺ compare() និង equals() ។ compare() method ធ្វើការប្រៀបធៀបធាតុពីតាមលំដាប់មានទំរង់ដូចខាងក្រោម :

int compare(Object *obj1*, Object *obj2*)

obj1 និង *obj2* គឺជា objects ដែលត្រូវធ្វើការប្រៀបធៀប ។ method នេះអោយតំលៃសូន្យបើសិនជា objects ស្មើគ្នា ។ វាអោយតំលៃវិជ្ជមានបើសិន *obj1* ធំជាង *obj2* ។ ផ្ទុយទៅវិញ វាអោយតំលៃអវិជ្ជមាន ។ method អាចប្រើ ClassCastException ប្រសិនបើប្រភេទនៃ objects មិនស៊ីគ្នាក្នុងការប្រៀបធៀប ។ ដោយសារការសរសេរជាន់លើ compare() method គេអាចជ្រើសយកវិធីផ្សេងដែល objects ត្រូវរៀបតាមលំដាប់ ។ ឧទាហរណ៍ដើម្បីរៀបតាម

លំដាប់បញ្ជ្រាស គេអាចបង្កើត comparator មួយដែលបញ្ជ្រួសលទ្ធផលនៃការប្រៀបធៀប ។

equals() method ធ្វើការផ្ទៀងផ្ទាត់ object មួយស្មើនឹង comparator ដែលកំពុងប្រើឬទេ :

boolean equals(Object *obj*)

obj គឺជា object ដែលត្រូវផ្ទៀងផ្ទាត់ចំពោះភាពស្មើគ្នា ។ method អោយតំលៃ true បើសិនជា *obj* និង object កំពុងប្រើគឺជា Comparator objects និងប្រើលំដាប់ដូចគ្នា ។ ផ្ទុយទៅវិញ វាអោយតំលៃ false ។

ការសរសេរលុបលើ equals() method គឺមិនសំខាន់ឡើយ ហើយ comparators ងាយបំផុតនឹងមិនធ្វើការសរសេរលុបលើដែរ ។

ឧទាហរណ៍ខាងក្រោមបង្ហាញអានុភាពនៃ comparator ប្រើតាមទំលាប់ ។ វាប្រើ method ឈ្មោះ compare() ដូចនេះវាប្រតិបត្តិតាមទិសបញ្ជ្រាសទៅនឹងសភាពធម្មតា ។ ហេតុនេះវាធ្វើអោយ tree set ត្រូវផ្ទុកតាមលំដាប់បញ្ជ្រាស ។

ពិនិត្យអោយល្អិតល្អន់ទៅលើ MyComp class ដែលប្រើ Comparator និងសរសេរជាន់លើ compare() ។ នៅក្នុង compare(), method របស់ String ឈ្មោះ compareTo() ធ្វើការប្រៀបធៀប strings ពីរ ។ ទោះជា យ៉ាងណាក៏ដោយ bStr (មិនមែន aStr) ដែលប្រើ compareTo() ។ នេះបណ្តាលអោយលទ្ធផលនៃការប្រៀបធៀបដែលត្រូវធ្វើការបញ្ជ្រាស ។

ចំពោះឧទាហរណ៍ជាក់ស្តែង កម្មវិធីខាងក្រោមនេះគឺជាកំណែថ្មីរបស់កម្មវិធី TreeMap ពីផ្នែកខាងដើមដែលផ្ទុកទឹកប្រាក់បញ្ជីរ ។ នៅក្នុងកំណែថ្មីមុននេះគណនីត្រូវរៀបតាមឈ្មោះ ក៏ប៉ុន្តែការរៀបតាមលំដាប់ ចាប់ផ្តើមដោយនាមខ្លួន ។ កម្មវិធីខាងក្រោមនេះរៀបលំដាប់នូវ accounts តាមនាមត្រកូល ។ ដើម្បីធ្វើអោយបានបែបនេះ វាប្រើ comparator ធ្វើការប្រៀបធៀបនាមត្រកូលរបស់ account និមួយៗ ។ លក្ខណៈនេះនាំមកលទ្ធផលតាម map ដែលត្រូវរៀបតាមនាមត្រកូល ។

comparator class ឈ្មោះ TComp ធ្វើការប្រៀបធៀប strings ពីរដែលផ្ទុកនាមខ្លួននិងនាមត្រកូល ។វាធ្វើបែបនេះដោយប្រៀបធៀបនាមត្រកូលមុនគេ ។ ដើម្បីធ្វើការប្រៀបធៀបនេះ វារក index នៃផ្នែកខាងចុងក្នុង string និមួយៗ រួចហើយធ្វើការប្រៀបធៀប substrings នៃធាតុនិមួយៗដែលចាប់ផ្តើមនៅត្រង់ចំណុចនោះ ។ នៅក្នុងករណីដែលនាមត្រកូលដូចគ្នា ពេលនោះនាមខ្លួនចាប់ផ្តើមធ្វើការប្រៀបធៀប ។លក្ខណៈនេះធ្វើអោយ tree map ដែលត្រូវរៀបតាមនាមត្រកូល ហើយជាមួយនាមត្រកូលដូចគ្នាត្រូវប្រៀបធៀបនាមខ្លួន ។ គេអាចឃើញពីលក្ខណៈនេះដោយ Ralph Smith នៅពីលើឈ្មោះ Tom Smith ក្នុងលទ្ធផល ។

៨. អំពី Arrays class

Java 2 បានបន្ថែម class ថ្មីមួយទៅក្នុង java.util គឺ Arrays class ។ class នេះផ្តល់ methods ផ្សេងៗដែលមានសារៈសំខាន់នៅពេលគេធ្វើការជាមួយនឹង arrays ។ ទោះបីជា methods ទាំងនេះមិនមែនជាផ្នែកមួយនៃពពួក collections framework ក៏ដោយ ប៉ុន្តែវាជាស្ពានចំលងចន្លោះរវាងពពួក collections និង arrays ។ method និមួយៗកំណត់ដោយ Arrays class នឹងលើកឡើងក្នុងពេលនេះ ។

asList() method អោយតំលៃជា List ដែលត្រូវបានស្គាល់ដោយ array ។ គេអាចនិយាយម្យ៉ាងទៀតថា list និង array បញ្ជាក់អោយទីតាំងដូចគ្នា ។ វាមានទំរង់ដូចខាងក្រោម :

static List asList(Object[] *array*)

ក្នុងនេះ *array* គឺជា array ដែលបានផ្ទុកទិន្នន័យ ។

binarySearch() method ប្រើ binary search ដើម្បីរកតំលៃដែលបានកំណត់ ។ method នេះត្រូវប្រើជាមួយ arrays ដែលបានរៀបទិន្នន័យតាមលំដាប់រួចហើយ ។ វាមានទំរង់ដូចខាងក្រោម :

static int binarySearch(byte[] *array*, byte *value*)

static int binarySearch(char[] *array*, char *value*)

static int binarySearch(double[] *array*,
double *value*)

static int binarySearch(float[] *array*, float *value*)

static int binarySearch(int[] *array*, int *value*)

static int binarySearch(long[] *array*, long *value*)

static int binarySearch(short[] *array*, short *value*)

static int binarySearch(Object[] *array*, Object *value*)

static int binarySearch(Object[] *array*, Object *value*,
Comparator *c*)

ក្នុងនេះ *array* គឺជា array ដែលត្រូវរុករក ហើយ *value* គឺជាតំលៃដែលដាក់ទីតាំង ។ ទំរង់ពីរចុងក្រោយប្រើ ClassCastException បើសិនជា *array* មានធាតុដែលមិនអាចប្រៀបធៀប (ឧទាហរណ៍ Double និង StringBuffer) ឬបើសិនជា *value* មិនត្រូវគ្នានឹងប្រភេទទិន្នន័យក្នុង array ។ ចំពោះទំរង់ចុងក្រោយ Comparator *c* ត្រូវបានប្រើសំរាប់កំនត់លំដាប់នៃធាតុក្នុង array ។ នៅគ្រប់ករណីទាំងអស់ បើសិនជា *value* មានក្នុង array នោះគេនឹងទទួលបាន index នៃធាតុ ។ ផ្ទុយទៅវិញគេនឹងទទួលបានតំលៃអវិជ្ជមាន ។

equals() method អោយតំលៃ true បើសិនជា array ពីរស្មើគ្នា។ ផ្ទុយទៅវិញ វានឹងអោយតំលៃ false ។ equals() method មានទំរង់ដូចខាងក្រោម :

static boolean equals(boolean *array1*[],
boolean *array2*[])

static boolean equals(byte *array1*[], byte *array2*[])

static boolean equals(char *array1*[], char *array2*[])

static boolean equals(double *array1*[],
double *array2*[])

static boolean equals(float *array1*[], float *array2*[])

static boolean equals(int *array1*[], int *array2*[])

static boolean equals(long *array1*[], long *array2*[])

static boolean equals(short *array1*[], short *array2*[])

static boolean equals(Object *array1*[],
Object *array2*[])

ក្នុងនេះ *array1* និង *array2* គឺជា array ពីរដែលត្រូវ

ធ្វើការប្រៀបធៀបដើម្បីរកភាពស្មើគ្នា ។

fill() method កំណត់តំលៃមួយទៅអោយធាតុទាំងអស់ ក្នុង array ។ គេអាចនិយាយម្យ៉ាងទៀតថា វាបំពេញអោយ array នូវតំលៃកំណត់មួយ ។ fill() method មានពីរប្រភេទ ។ ប្រភេទទី ១ មានទំរង់ខាងក្រោមសំរាប់បំពេញអោយ array ទាំងមូល :

static void fill(boolean *array*[], boolean *value*)

static void fill(byte *array*[], byte *value*)

static void fill(char *array*[], char *value*)

static void fill(double *array*[], double *value*)

static void fill(foat *array*[], float *value*)

static void fill(int *array*[], int *value*)

static void fill(long *array*[],long *value*)

static void fill(short *array*[], short *value*)

static void fill(Object *array*[], Object *value*)

ក្នុងនេះ *value* ត្រូវកំណត់តំលៃទៅអោយគ្រប់ធាតុទាំងអស់ក្នុង *array* ។

ប្រភេទទី 2 fill() method កំណត់តំលៃមួយទៅអោយ array តូចៗ។ វាមានទំរង់របស់វាមានដូចខាងក្រោម:

static void fill(boolean *array*[], int *start*, int *end*, boolean *value*)
static void fill(byte *array*[], int *start*, int *end*, byte *value*)
static void fill(char *array*[], int *start*, int *end*, char *value*)
static void fill(double *array*[], int *start*, int *end*, double *value*)
static void fill(foat *array*[], int *start*, int *end*, float *value*)

static void fill(int *array*[], int *start*, int *end*, int *value*)
static void fill(long *array*[], int *start*, int *end*, long *value*)
static void fill(short *array*[], int *start*, int *end*, short *value*)
static void fill(Object *array*[], int *start*, int *end*, Object *value*)

ក្នុងនេះ *value* ត្រូវកំណត់តំលៃទៅអោយធាតុក្នុង *array* ពីទីតាំង *start* ទៅទីតាំង *end*–1 ។ methods ទាំងអស់នេះ អាចប្រើ exception ប្រភេទ IllegalArgumentException នៅពេលដែល *start* ធំជាង *end* ឬក៏

ArrayIndexOutOfBoundsException កាលណា *start* ឬ *end* ស្ថិតនៅក្រៅដែនកំណត់ index ។

sort() method តំរៀបទិន្នន័យក្នុង array ដូច្នេះវាតំរៀបនៅក្នុងលំដាប់កើនឡើង ។ sort() method មានពីរប្រភេទ ។

sort() method ប្រភេទទីមួយរៀបទិន្នន័យក្នុង array ទាំងមូល ។

static void sort(byte *array*[], int *start*, int *end*)
static void sort(char *array*[], int *start*, int *end*)

static void sort(double *array*[], int *start*, int *end*)

static void sort(foat *array*[], int *start*, int *end*)

static void sort(int *array*[], int *start*, int *end*)

static void sort(long *array*[], int *start*, int *end*)

static void sort(short *array*[], int *start*, int *end*)

static void sort(Object *array*[], int *start*, int *end*)

static void sort(Object *array*[], int *start*, int *end*, Comparator *c*)

ក្នុងនេះចន្លោះពី *start* ទៅ *end*–1 ក្នុង array នឹងត្រូវរៀបតាមលំដាប់ ។ ទំរង់ចុងក្រោយបង្អស់ *c* គឺជា Comparator ដែលត្រូវបានប្រើសំរាប់រៀបធាតុនៃ *array* តាមលំដាប់ ។

methods ទាំងអស់នេះអាចប្រើ IllegalArgumentException កាលណា *start* ធំជាង *end* ឬ ArrayIndexOutOfBoundsException កាល ណា *start* ឬ *end* ស្ថិតនៅក្រៅដែនកំណត់ index ។ ទំរង់ទាំងពីរចុងក្រោយបង្ហូស់អាចប្រើ ClassCastException កាលណាធាតុនៃ array ដែលត្រូវរៀបតាមលំដាប់នោះមិនបានធ្វើការប្រៀបធៀប ។

កម្មវិធីខាងក្រោមនេះបង្ហាញពីរបៀបប្រើ methods មួយចំនួនរបស់ Arrays class :

៩. អំពី Vector class

Vector អនុវត្តនូវ array ដែលមានលក្ខណៈ dynamic ។ វាស្រដៀងទៅនឹង ArrayList ដែរ ក៏ប៉ុន្តែវាមានភាពខុសគ្នាពីរគឺ Vector មានលក្ខណៈ synchronized ហើយនិងមាន methods ជាច្រើន ដែលមិនមែនជាចំណែកនៃពួក collections ។ ជាមួយនឹង Java 2 នេះ Vector ត្រូវបានធ្វើអោយមាន ការទទួលលក្ខណៈពី AbstractList និងអនុវត្តតាម List interface ហេតុនេះវាមានលក្ខណៈត្រូវគ្នាជាមួយនឹង collections យ៉ាងពេញលេញ ។

ទំរង់ constructors របស់ Vector គឺ:

Vector()

Vector(int *size*)

Vector(int *size*, int *incr*)

Vector(Collection *c*)

ទំរង់ constructor ទីមួយបង្កើតនូវ vector ធម្មតា ដែលមានការកំណត់តំលៃទំហំដំបូងគឺ 10 ។ ទំរង់ទីពីរបង្កើតនូវ vector មួយដែលមានចំណុះដំបូងទៅតាមការកំណត់ *size* ហើយកំណើនរបស់វាត្រូវបានកំណត់ដោយ *incr* ។

កំណើនបញ្ជាក់អោយចំនួនធាតុដើម្បីដាក់ទៅទីតាំងរៀងរាល់ពេលដែល vector មួយប្តូរទំហំកើនឡើង។ ទំរង់ទី ៤បង្កើត vector មួយដែលមានធាតុនៃ collection *c* ។ constructor នេះត្រូវបានបន្ថែមឡើងដោយ Java 2 ។

ក្រប់ vectors ទាំងអស់ចាប់ផ្តើមឡើងដោយមានចំណុះតំលៃដំបូង។ បន្ទាប់មកចំណុះតំលៃដំបូងនេះបានជួបប្រទះ ហើយនៅពេលក្រោយគេចង់ផ្ទុក object មួយក្នុង vector នោះ vector នឹងបង្កើតទីតាំងសំរាប់ object នោះដោយបូកបន្ថែមលើទំហំសំរាប់ object បញ្ជូលដោយស្វ័យប្រវត្ត។

តាមរយៈបង្កើត ទីតាំងលើសពីសេចក្តីតំរូវការ memory នោះ vector បន្ថយចំនួនបង្កើតទីតាំងដែលត្រូវការប្រើ ។ ការបន្ថយនេះមានសារៈសំខាន់ព្រោះការបង្កើតទីតាំងគឺចំណាយពេលច្រើន ។ បរិមាណនៃទំហំបន្ថែមបានបង្កើតទីតាំងក្នុងកំឡុងពេលបង្កើតទីតាំងជាថ្មីត្រូវបានកំនត់តាមរយៈកំណើនដែលយើងបានបញ្ជាក់ពេលគេបង្កើត vector ។ បើសិនជាគេមិនបញ្ជាក់នូវចំនួនកំណើននោះទំហំ vector នឹងកើនឡើងទ្វេតាមរយៈដំណើរបង្កើតទីតាំងនិមួយៗ ។

Vector បានបង្កើត data members ទាំងនេះដែលមាន

លក្ខណៈ protected :

int capacityIncrement;

int elementCount;

Object elementData[];

តំលៃកំណើនត្រូវផ្ទុកក្នុង capacityIncrement ។ ចំនួន

ធាតុក្នុង vector ត្រូវផ្ទុកក្នុង elementCount ។ array

ដែលផ្ទុក vector ត្រូវដាក់ក្នុង elementData ។

methods របស់ Vector class ដែលអាចប្រើបានបន្ថែមទៅលើ methods ដែលមានក្នុងពពួក collections មានដូចក្នុងតារាងខាងក្រោមនេះ ។

ដោយសារតែ Vector អនុវត្តតាម List interface គេអាចប្រើ vector ដូចទៅនឹង ArrayList object ដែរ ។ ឧទាហរណ៍ បន្ទាប់ពីបង្កើត object នៃ Vector រួចហើយគេអាចបញ្ចូលធាតុមួយទៅអោយវាដោយប្រើ addElement() ។ ដើម្បីទទួលបានធាតុនៅត្រង់ទីតាំងដែលបានកំណត់ គេប្រើ elementAt() ។

ដើម្បីទទួលបានធាតុទី១ក្នុង vector គេប្រើ firstElement() ។ ដើម្បីទទួលបានធាតុចុងក្រោយក្នុង vector គេប្រើ lastElement() ។ គេអាចទទួលបាន index នៃធាតុមួយតាមរយៈការប្រើ indexOf() និង lastIndexOf() ។ ដើម្បីលុបធាតុមួយ គេប្រើ removeElement() ឬ removeElementAt() ។

ឈ្មោះ Methods	អធិប្បាយ
final void addElement(Object *element*) object	បញ្ចាក់តាមរយៈ *element* ដែលត្រូវបន្ថែមទៅលើ vector ។
final int capacity()	អោយតំលៃជាចំណុះនៃ vector ។
Object clone()	អោយតំលៃស្ទួននៃ vector ដែលកំពុងប្រើ ។
final boolean contains(Object *element*)	អោយតំលៃ true បើសិនជា *element* បានផ្ទុកដោយ vector ហើយអោយតំលៃ false បើសិនវាមិនបានផ្ទុក ។

final void copyInto(Object *array*[]) ធាតុដែលបាន ផ្ទុកក្នុង vector កំពុងប្រើត្រូវចំលងទៅ ក្នុង array ដែលបានកំណត់ដោយ *array* ។

final Object elementAt(int *index*) អោយតំលៃ ជាធាតុត្រង់ទីតាំងកំណត់ដោយ *index* ។

final Enumeration elements() អោយតំលៃជារបាប់ នៃធាតុក្នុង vector ។

final void ensureCapacity(int *size*) កំណត់ចំណុះ អប្បបរមានៃ vector ទៅអោយ *size* ។

final Object firstElement() អោយតំលៃជាធាតុ

ទីមួយក្នុង vector ។

final int indexOf(Object *element*) អោយតំលៃ

ជា index នៃការកើតឡើងរបស់ *element* ។ បើសិនជា

object មិនមានក្នុង vector នោះវានឹង អោយតំលៃ –1 ។

final int indexOf(Object *element*, int *start*)

អោយតំលៃជា index នៃការកើតឡើងរបស់ *element*

នៅត្រង់ឬបន្ទាប់ *start* ។ បើសិនជា object មិនមានក្នុង

vector ពេលនោះវាអោយតំលៃ –1 ។

final void insertElement(Object *element*, int *index*) បញ្ចូល *element* ទៅអោយ vector នៅត្រង់ទីតាំងដែលបានកំណត់ដោយ *index* ។

final boolean isEmpty() អោយតំលៃ true បើសិនជា vector ពុំមានធាតុសោះ ហើយអោយតំលៃ false បើសិនជាវាមានធាតុមួយឬច្រើន ។

final Object lastElement() អោយតំលៃជាធាតុ ខាងចុងបង្អស់ក្នុង vector ។

final int lastIndexOf(Object *element*) អោយ

តំលៃជា index នៃការកើតឡើងផ្នែកខាងចុង

របស់ *element* ។ បើសិនជា object មិនមានក្នុង

vector នោះវា នឹងអោយតំលៃ –1 ។

final int lastIndexOf(Object *element*, int *start*)

អោយតំលៃជា index នៃការកើតឡើងផ្នែកខាង

ចុងនៃ *element* មុន *start* ។ បើសិនជា

object មិនមានក្នុង vector ពេលនោះ

វានឹងអោយតំលៃ –1 ។

final void removeAllElements() ធ្វើអោយ vector គ្មានធាតុ ។ បន្ទាប់ពី method នេះប្រតិបត្តិការ នោះទំហំរបស់ vector ស្មើនឹងសូន្យ ។

final boolean removeElement(Object *element*)

បំបាត់ *element* ពី vector ។ បើសិនជាវាមានច្រើនជាងមួយ object ដែលបានកំណត់ក្នុង vector ពេលនោះ វាត្រូវបានលុបមុនគេ ។ វាអោយតំលៃ true បើសិនជាបានជោគជ័យ និងអោយតំលៃ false កាលណា object មិនមាន ។

final void removeElementAt(int *index*) បំបាត់ធាតុនៅត្រង់ទីតាំងដែលបានកំណត់ដោយ *index* ។

final void setElementAt(Object *element*, int *index*) ទីតាំងដែលបានកំណត់ដោយ *index* ត្រូវកំណត់ទៅអោយ *element* ។

final void setSize(int size) កំណត់ចំនួនធាតុក្នុង vector ទៅអោយ *size* ។ បើសិនជាទំហំថ្មីតូចជាងទំហំចាស់ ពេលនោះធាតុត្រូវបាត់បង់ ។ បើសិនជាទំហំថ្មីធំជាងទំហំចាស់ ពេលនោះធាតុ null ត្រូវបានដាក់បញ្ចូល ។

final int size() អោយតំលៃជាចំនួនធាតុកំពុងមានក្នុង vector ។

String toString() អោយតំលៃជា string ដែលសមមូលនឹង vector ។

final void trimToSize() កំនត់ចំនុះរបស់ vector ស្មើនឹងចំនួនធាតុដែលវាកំពុងផ្ទុក ។

កម្មវិធីខាងក្រោមនេះប្រើ vector មួយសំរាប់ផ្ទុកប្រភេទផ្សេងៗរបស់ object ជាលេខ។ វាបង្ហាញនូវ methods ជាច្រើនដែលកំនត់ដោយ Vector class។ វាក៏បានបង្ហាញផងដែរនូវការប្រើ Enumeration interface។

នៅក្នុង Java 2 គេអាចប្រើ iterators ជាមួយនឹង Vector បានដែរ។ គេប្រើ iterator ជំនួសអោយការប្រើរបាប់ (enumeration)។ ចូរសង្កេតឧទាហរណ៍ខាងក្រោមនេះ:

```
// use an iterator to display contents
Iterator vItr = v.iterator();
System.out.println("\nElements in vector:");
while(vItr.hasNext())
    System.out.print(vItr.next() + " ");
System.out.println();
```

១០. អំពី Hashtable class

Hashtable class ស្ថិតក្នុង java.util ហើយជា subclass មួយដែលបានទទួលលក្ខណៈពី abstract class ឈ្មោះ Dictionary ។ ទោះជាយ៉ាងណាក៏ដោយ Java 2 បានបង្កើតជាថ្មីនូវ Hashtable ដោយអោយវាអនុវត្តតាម Map interface ។ ដូច្នេះ Hashtable បានបញ្ចូលទៅក្នុងពួក collections ។

Hashtable ផ្ទុកតំលៃ key/value ជាគូនៅក្នុងតារាង "hash" ។ នៅពេលប្រើ Hashtable គេបញ្ជាក់នូវ object ដែលត្រូវប្រើជា key និងតំលៃដែលគេចង់ភ្ជាប់ទៅនឹង key នោះ ។ ពេលនោះ key ត្រូវធ្វើអោយទៅជា hash ហើយ hash code ដែលជាលទ្ធផលត្រូវបានប្រើជា index ដែលតំលៃនោះផ្ទុកក្នុងតារាង ។

hash table អាចផ្ទុកតែ objects ប៉ុណ្ណោះដែលសរសេរលុបលើ hashCode() និង equals() method ហើយត្រូវបានកំណត់ដោយ Object ។ hashCode() method

ត្រូវគណនា និងអោយតំលៃជា hash code សំរាប់ object ។ តាមពិត equals() ធ្វើការប្រៀបធៀប objects ពីរ ។ Java ផ្តល់លក្ខណៈ ងាយស្រួលជាច្រើនដោយវាមាន classes ដែលមានស្រាប់ប្រើនូវ hashCode() method ។ ឧទាហរណ៏ ប្រភេទទិន្នន័យភាគច្រើនប្រើ String object ជា key ។ String ប្រើទាំង hashCode() និង equals() ។

ទំរង់ constructors របស់ Hashtable មាន:

Hashtable()
Hashtable(int *size*)

Hashtable(int *size*, float *fillRatio*)

Hashtable(Map *m*)

ទំរង់ constructor ទីមួយគឺជា default constructor ។ ទំរង់ constructor ទីពីរបង្កើត hash table មួយដែលមានទំហំតំលៃដំបូងកំណត់តាមរយៈ *size* ។ ទំរង់ constructor ទីបីបង្កើតនូវ hash table មួយដែលមានទំហំតំលៃដំបូងកំណត់តាមរយៈ *size* និងមានចំនួនសមាមាត្រដែលត្រូវបំពេញ ។

ចំនួនសមាមាត្រនេះត្រូវស្ថិតក្នុងតំលៃចន្លោះ 0.0 និង 1.0 ហើយ វាកំណត់នូវរបៀប hash table ពេញមុននឹងវាត្រូវប្តូរទំហំ ។ ជាពិសេសនៅពេលចំនួនធាតុធំជាងចំណុះនៃ hash table គុណ នឹងលេខសមាមាត្រដែលត្រូវបំពេញរបស់វា ពេលនោះ hash table ត្រូវពង្រីក ។ បើសិនគេមិនបានបញ្ជាក់ចំនួនសមាមាត្រទេ នោះវានឹងប្រើ 0.75 ។ ទំរង់ constructor ចុងក្រោយបង្អស់ បង្កើតនូវ hash table មួយដែលត្រូវកំណត់តំលៃដំបូងដោយ ធាតុជា *m* ។ ផលគុណកត្តាធម្មតាគឺ 0.75 ដែលត្រូវបានគេប្រើ ។ ទំរង់ constructor ទីបួននេះត្រូវបានបញ្ចូល ដោយ Java 2 ។

លើសពីនេះ ក្រៅពី methods ដែលបានកំណត់ដោយ Map interface នោះ វានៅមាន methods មួយចំនួនទៀតដែល Hashtable មានហើយបានរាយឈ្មោះដូចខាងក្រោមនេះ :

ឈ្មោះ Methods	អធិប្បាយ
void clear()	ដាក់សាជាថ្មី និងធ្វើអោយ hash table ទទេគ្មានអ្វីទាំងអស់ ។
Object clone()	អោយតំលៃ object ដែលកំពុងប្រើចំនួនពីរ ។

boolean contains(Object *value*) អោយតំលៃ true

បើមានតំលៃមួយចំនួនស្មើនឹង *value* ដែលមាន ហើយក្នុង

hash table ។ អោយតំលៃ false បើសិនជា

តំលៃនោះមិនមានទេ ។

boolean containsKey(Object *key*) អោយតំលៃ true

បើមានតំលៃមួយចំនួនស្មើនឹង *key* ដែលមានហើយ ក្នុង

hash table ។ អោយតំលៃ false បើសិនជា key

នោះមិនមានទេ ។

boolean containsValue(Object *value*) អោយតំលៃ true បើមានតំលៃមួយចំនួនស្មើនឹង *value* ដែលមានហើយក្នុង hash table ។ អោយតំលៃ false

បើសិនជាតំលៃនោះមិនមានទេ ។

Enumeration elements() អោយតំលៃជាចំនួនរាប់

នៃតំលៃដែលបានផ្ទុកក្នុង hash table ។

Object get(Object *key*) អោយតំលៃជា object ដែលបានផ្ទុកតំលៃទាក់ទងនឹង *key* ។ បើសិនជា *key* មិនមានក្នុង hash table ទេនោះវានឹងអោយ null object ។

boolean isEmpty() អោយតំលៃ true បើសិនជា hash table ពុំមាន key សោះ ហើយអោយតំលៃ false បើសិនជាវាមាន key មួយយ៉ាងតិច ។

Enumeration keys() អោយតំលៃជាចំនួនរបាប់នៃ key ដែលបានផ្ទុកក្នុង hash table ។

Object put(Object *key*, Object *value*) បញ្ចូល key មួយនិងតំលៃមួយទៅក្នុង hash table ។ អោយតំលៃ null បើសិនជា *key* មិនទាន់មានក្នុង hash table ហើយអោយតំលៃមុនដែលមានទំនាក់ទំនងជាមួយនឹង *key* បើសិនជា *key* មានក្នុង hash table រួចហើយ ។

void rehash() បង្កើនទំហំរបស់ hash table ហើយរៀបដាក់ជាថ្មីគ្រប់ keys ទាំងអស់របស់វាក្នុង hash table ។

Object remove(Object *key*) លុបបំបាត់ហើយ និងតំលៃរបស់វា។ អោយតំលៃដែលទាក់ទងទៅនឹង *key* ។ បើសិនជា *key* មិនមានក្នុង hash table ទេនោះ វានឹងអោយ null object ។

int size() អោយតំលៃជាចំនួននៃការបញ្ចូលក្នុង hash table ។

String toString() អោយតំលៃជា string ដែលសមមូលនឹង hash table ។

កម្មវិធីខាងក្រោមនេះបង្ហាញពីការប្រើ Hashtable ដើម្បីផ្ទុកឈ្មោះរបស់អ្នកបញ្ជីរប្រាក់នៃធនាគារ ហើយនិងទឹកប្រាក់កំពុងមានរបស់ពួកគេ ។

ចំណុចសំខាន់មួយគឺដូចទៅនឹង map class ដែរ Hashtable មិនអាចប្រើ iterators បានដោយផ្ទាល់ឡើយ ។ ហេតុនេះកម្មវិធីមុននេះប្រើ enumeration ដើម្បីបង្ហាញតំលៃនៃចំនួនទឹកប្រាក់ ។ ទោះជាយ៉ាងណាក៏ដោយ គេអាចទទួលបាននូវទិដ្ឋភាពនៃ hash table ដែលអាចអោយគេប្រើ iterators ។

ដើម្បីធ្វើដូច្នោះបាន គេប្រើ method មួយក្នុងចំណោម methods របស់ពពួក collection ដែលកំណត់ដោយ Map ដូចជា entrySet() ឬ keySet() ។ ឧទាហរណ៍ គេអាចទទួលបាននូវទិដ្ឋភាពនៃ keys ហើយអោយវាធ្វើសកម្មភាពឆ្លងកាត់ keys ។ ចូរសង្កេតនូវវិធីនេះក្នុងកម្មវិធីខាងក្រោម:

សំនួរឆ្លើយលំហាត់

១ - ចូរសរសេរកម្មវិធីមួយដែលអាចអោយគេប្រើ method របស់ String ឈ្មោះ compareTo ធ្វើការប្រៀបធៀប strings ពីរដែលបញ្ចូលដោយអ្នកប្រើប្រាស់ ។ លទ្ធផលរបស់វាអាចជា string ទីមួយមានតំលៃតូចជាង ឬ ដូចគ្នា ឬធំជាង string ទីពីរ ។

- ២ - ចូរសរសេរកម្មវិធីមួយដែលអាចអោយគេប្រើ method របស់ String ឈ្មោះ regionMatches ដើម្បី ធ្វើការប្រៀបធៀប strings ពីរដែលបញ្ចូលដោយអ្នកប្រើប្រាស់ ។ កម្មវិធីត្រូវតែបញ្ចូលចំនួនតួអក្សរ ដែលត្រូវធ្វើការប្រៀបធៀប ហើយចាប់ផ្តើមនៅត្រង់ index នៃការប្រៀបធៀប ។ កម្មវិធីត្រូវតែប្រាប់អោយដឹង រ៉ាតើ strings ដូចគ្នា ឬមិនដូចគ្នា ។ ហើយនៅពេលធ្វើការប្រៀបធៀប រ៉ាមិនប្រកាន់តួអក្សរតូចធំឡើយ ។

៣ - ចូរសរសេរកម្មវិធីមួយដែលអាចអោយគេបញ្ចូលអក្សរមួយឃ្លា ហើយពេលនោះលទ្ធផលរបស់វានឹងបង្ហាញអក្សរនោះពីរជួរ ដែលជួរទីមួយជាអក្សរធំទាំងអស់ និងជួរទីពីរជាអក្សរតូចទាំងអស់ ។

៤ - ចូរសរសេរកម្មវិធីមួយដែលអាចអោយគេបញ្ចូលតួអក្សរមួយឃ្លា រួចហើយបញ្ចូលតួអក្សរមួយតួដែលត្រូវរកនៅក្នុងអក្សរមួយឃ្លានោះដោយប្រើ method របស់ String ឈ្មោះ indexOf() ដើម្បីរកលេខត្រង់ទីតាំងនៃការកើតឡើងរបស់តួអក្សរមានក្នុងឃ្លានោះ ។

៥ - តើ trim() method របស់ String មានមុខងារសំរាប់ធ្វើអ្វី? ចូរសរសេរ code បង្ហាញពីការប្រើវា ។

៦ - តើ code ខាងក្រោមនេះនឹងអោយលទ្ធផលអ្វី កាលណាគេ compile និង run វា?

```
class RefEq {
    public static void main(String[] args) {
        String s = "ab" + "12";
        String t = "ab" + 12;
        String u = new String("ab12");
        System.out.println((s==t) + " " + (s==u));
    }
}
```