Tipuri generice, Colecții în Java

Generic types, Collections in Java

Obiective:

- înțelegerea mecanismelor tipurilor generice; clase generice; interfețe generice; metode generice;
- utilizarea principalelor tipuri de colecție;

Objectives:

- understanding the generic types mechanisms; generic classes; generic interfaces; generic methods;
- using the main collection types;

Tipuri generice

Mecanismul tipurilor generice permite tipurilor de date să fie parametri la definirea claselor, metodelor și interfețelor.

- Avantaje:
  - robustețea codului (verificări de tip la compilare)
  - eliminarea conversiilor de tip cast
  - algoritmi mai generali

Sintaxa:

class name<T1, T2, ..., Tn> { / ... */ }*
interface name<T1, T2, ..., Tn> { / ... */ }*
[specificator_acces] [static] <K, V> tip_returnat name(generic_type <K, V> p1, generic_type<K, V> p2) { /...*/}*

Convenții de numire a tipurilor generice

- E - Element (folosit extensiv de Java Collections Framework)
- K - Key
- N - Number
- T - Type
- V - Value

Colecții Java

Java pune la dispoziţia programatorilor o serie de clase/interfeţe cu ajutorul cărora se pot gestiona liste sau colecţii de obiecte care aparţin unor alte clase. În acest caz numărul de elemente nu este cunoscut de la început.

Definiție: *o colecţie este un grup de obiecte, ordonat sau nu şi care poate conține valori repetate sau nu*. Există colecţii care pot conţine doar un anumit tip de elemente (de exemplu instanțe ale unei singure clase) şi există colecţii care pot stoca obiecte de diverse tipuri. De asemenea, anumite colecţii prevăd posibilitatea introducerii de elemente *null*, iar altele nu.

Colecţiile Java oferă mecanisme de inserare de elemente în colecţii, de căutare a anumitor componente, de ordonare, etc.

Interfaţa *Collection*

Această interfaţă stă la baza tuturor interfeţelor şi a claselor care lucrează cu colecţii în Java.

Interfeţe care moştenesc interfaţa *Collection*: BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, TransferQueue<E>

Clase care utilizează (implementează) interfaţa *Collection*: AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentHashMap.KeySetView, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

Metodele declarate în interfaţa *Collection* sunt prezentate mai jos.

- *Boolean add(Object o)*
  => adauga un element în colecţie
- *boolean addAll(Collection c)*
  => adauga toate elementele din colecţia primită ca parametru.
- *void clear()*
  => şterge toate elementele din colecţie
- *boolean contains(Object o)*
  => returnează o valoare booleana în funcţie de existenţa obiectului căutat
- *containsAll(Collection c)*
  => returnează o valoare booleana în funcţie de existenţa unei întregi colecţii (parametrul primit)
- *boolean equals(Object o)*
  => compara 2 obiecte de tip colecţie
- *int hashCode()*
  => returnează valoarea *hash* a colecţiei
- *boolean isEmpty()*
  => testează dacă colecţia are sau nu elemente
- *Iterator iterator()*
  => returnează un iterator pentru elementele colecţiei
- *boolean remove(Object o)*
  => elimina (dacă exista) din colecţie elementul primit ca parametru
- *boolean removeAll(Collection c)*
  => elimina din colecţie toate elementele din colecţia primită ca parametru
- *boolean retainAll(Collection c)*
  => păstrează doar elementele care sunt conţinute de colecţia primită ca parametru
- *int size()*
  => returnează numărul de elemente din colecţie
- *Object[] toArray()*
  => returnează un şir de obiecte populat cu elementele din colecţie

Interfaţa *List*

Pune la dispoziţie un mecanism de control al unei colecţii ordonate de obiecte. Utilizatorul are controlul exact al poziţiei de inserare a unui element şi poate accesa orice obiect stocat prin intermediul indexului sau de tip int.

Listele accepta duplicate de obiecte.

În ceea ce priveşte metodele conţinute de interfaţă *List*, aceasta are specificaţii diferite sau în plus faţă de *Collection* pentru următoarele categorii de metode:

- metode pentru accesul indexat la elementele listei

- metode pentru căutarea unui element în listă
- metode pentru inserarea şi eliminarea elementelor din listă
- metoda de testare a egalităţii între 2 obiecte de tip lista

Clase care implementează interfaţa *List*:

AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector

Interfaţa *Set/SortedSet*

Un set este o colecţie fără elemente duplicate.

Faţă de interfaţa *Collection*, exista diferenţe sau metode noi în ceea ce priveşte:

- constructorii
- metodele de adăugare de elemente în set
- medota de calculare a valorii *hash*
- metoda de testare a egalităţii între 2 seturi

Elementele de tip *SortedSet* trebuie să implementeze interfaţa *Comparable* pentru a permite traversarea şi compararea elementelor.

Clase care implementează interfaţa *Set/SortedSet*

AbstractSet, ConcurrentHashMap.KeySetView, ConcurrentSkipListSet, CopyOnWriteArraySet, EnumSet, HashSet, JobStateReasons, LinkedHashSet, TreeSet

Interfaţa *Map/SortedMap*

Un obiect de acest tip face asocieri intre chei şi valori/obiecte. Cheile sunt unice. O cheie poate fi asociată unei singure valori.

Sunt puse la dispoziţie 3 modalităţi de a parcurge elementele stocate:

- în funcţie de chei
- în funcţie de valori
- în funcţie de asocierea dintre chei şi valori

Ordinea de parcurgere a elementelor depinde de modurile menţionate mai sus. Anumite clase care implementează această interfaţă specifică modul de parcurgere, altele nu. Există de asemenea clase care folosesc această interfaţă şi care impun restricţii asupra tipurilor de chei sau a tipurilor obiectelor stocate.

Interfaţa *SortedMap* implementează interfaţa *Map* permiţând şi comparări.

Clase care implementează interfaţa *Map/SortedMap*

AbstractMap, Attributes, AuthProvider, ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, HashMap, Hashtable, IdentityHashMap, LinkedHashMap, PrinterStateReasons, Properties, Provider, RenderingHints, SimpleBindings, TabularDataSupport, TreeMap, UIDefaults, WeakHashMap

Lucru individual

1. Creați o interfață denumită *Generator<T>* cu singura metodă *next(T var)*. Implementați interfața pentru a permite generarea valorilor următoare în cazul aplicării asupra unor variabile de tipuri de date concrete (*Integer*, *Character*, etc.). Instanțierea clasei se va face în cadrul metodei main() care va fi poziționată într-o clasă distinctă.

==========

2. Creați o clasă *Calculator* ce are implementate metode de adunare, scădere, înmulțire și împărțire. Metodele vor avea va intrare tipuri de date generice și vor returna rezultatul conform acestor date. De exemplu, suma a doi întregi va returna tot un întreg, pentru două numere de tip float se va returna tot un float. Aplicarea metodelor de adunare și scădere este permisă și pe variabile de tip *String*, restul operațiilor fiind interzise pt. acest tip de date (mesaj de eroare).

Scrieți aceeași clasă folosind supraîncărcarea metodelor.

==========

3. Implementați o aplicație în cadrul căreia să aveți o clasă SetterGetter generică ce permite setarea și returnarea valorilor atributelor pentru mai multe tipuri de obiecte. De exemplu, având clasele Copil, Adult și Pensionar, să se poată seta și returna numele și vârsta acestora. Creați colecții cu intrări unice de obiecte de tip Copil, Adult, Pensionar pe care să le populați cu date citite din consolă. Afișați datele preluate în diverite moduri.

==========

4. Implementati o clasa numita *UserFile* (denumire, extensie, tip, marime in bytes, constructori, mutatori si accesori). Tipurile de fisiere sunt predefinite si stocate intr-un obiect *Hashtable* (de ex. „imagine” =>0, „text” =>1, „aplicatie” =>2, etc.). Creați un șir de obiecte instanțiate din această clasă și citiți de la tastatură datele aferente. Afișați toate extensiile specifice tipurilor de fișiere predefinite. Ordonați lista de fișiere în funcție de mărime si afișați rezultatul.

==========

5. Creați o clasă generică denumită *BibliotecaVirtuala* a cărei singur atribut este *numărTotalIntrari*, iar ca metode permite setarea și returnarea unei intrări. Tipurile de intrări din bibliotecă pot fi *Carte*, *Articol*, *ResursăMedia*, *Revistă* și *Manual*. Implementați clasele aferente acestor tipuri de date.

În metoda main() creați o variabilă de tip *SortedSet<BibliotecaVirtuala>* care să rețină toate intrările din bibliotecă. Utilizați metodele de adăugare, adăugare multiplă, returnare a unei intrări și verificarea existenței unei intrări în bibliotecă.

Individual work

1. Create an interface called *Generator<T>* with a single method *next(T var)*. Implement the interface so that you can generate the following values when applying it to certain data types (Integer, Character, etc.). The class will be instantiated in the *main()* method, located in a separate class.

==========

2. Write a class called *Calculator* which has the methods to do addition, subtraction, multiplication and division. The methods will take generic input variables and will return the corresponding type. For example, the sum of two integers should return an integer, and for floats it should return a float. Same for division. Adding and subtracting is allowed for *String* variables as well, but the multiplication and division will print an error message.

Write the same class, but use method overloading.

==========

3. Build an application which contains a generic class *SetterGetter* which allows the user to *set()* and *get()* the attribute values for different types of objects. For example, given the classes *Kid, Adult* and *Retired*, enable the class to set and get the names and ages of the associated objects. Create collections with unique entries of type *Kid, Adult and Retired*, and which are populated with data read from the console. Print the read data using different methods.

==========

4. Implement a class called *UserFile* (name, extension, type, size in bytes, constructors, mutators, accesors). The types of files are predefined and stored in a *Hashtable* object (for example "image"->0, "text"->1, "application"->2, etc.) Create a list of objects from this class and read from the keyboard the associated info. Print all the specific extensions of the predefined file types. Order the file list based on size and print the result.

==========

5. Create a generic class called *VirtualLibrary* with a single attribute, *totalNumberOfEntries*, and with methods which enables the user to set and return an entry. The types of entries are *Book, Article, MediaResource, Magazine and Manual*. Implement the specific classes for each type of entry.

In the *main()* method create a *SortedSet<VirtualLibrary>* variable which maintains all the library entries. Use the methods to add, add multiple, return a specific entry and check if an entry exists in the library.