PAU CASESNOVES
Centre Integrat de Formació Professional

Desenvolupament d'aplicacions web

Programació

UT 8: Contenidors de dades.

Joan Pons Tugores

Índex de continguts

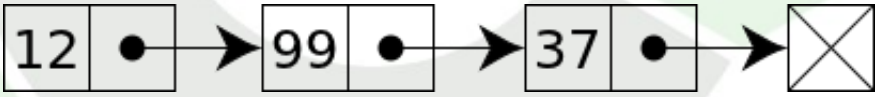
Contenidors de dades

Entendrem per contenidor de dades qualsevol tipus de dada que ens permeti emmagatzemar-hi un conjunt d'elements, evidentment recuperar-los, esborrar-los, ...

L'exemple més senzill de contenidors és un array. En crear un array estam creant una estructura que ens permet totes les accions que hem enumerat abans.

Nosaltres podem definir qualsevol contenidor de dades creant una classe que ens permeti emmagatzemar-les i que ens ofereixi una sèrie de mètodes per treballar amb elles.

Podem utilitzar un array per guardar les dades, però tenim altres possibilitats com poden ser les llistes encadenades. Cada element d'una llista encadenada està format per al manco dues parts: una que conté la informació i una altra que enllaça amb el següent element.



Utilitzant aquests eines, els arrays i les llistes encadenades, podem definir altres estructures de dades, per exemple arbres, coes, piles, conjunts, mapes, ...

Llistes

Una llista es defineix com una seqüència d'elements, sense cap condicionant específic. Un array es pot considerar una llista.

Algunes de les classes de l'API que defineixen una llista

- ArrayList: Contenidor que es pot utilitzar tant com un array com com una llista. Ofereix operacions com add (afegir un element), get (recuperar-lo), toArray(torna un array amb els elements que conté), ...
- LinkedList: Llista doblement encadenada, cada element apunta al següent i a l'anterior. Inclou totes les operacions definides per a llistes, piles i coes.

Recórrer una llista

Els elements d'una llista són accessibles per índex, per tant es poden recórrer utilitzant un for clàssic

```
ArrayList<Alumne> llista = new ArrayList<>();

    ...

    for(int i=0; i<llista.size(); i++){

        System.out.println(llista.get(i));

    }
```

La forma més recomanable però, és utilitzar una altra versió del for. És la més còmoda quan no necessitam manipular índexs:

```
ArrayList<Alumne> llista = new ArrayList<>();

    ...

    for(Alumne alumne:llista){

        System.out.println(alumne);

    }
```

Dins del *for* es declara una variable del tipus dels elements de la llista i es separa amb *:* de la llista. A cada iteració tenim un element de la llista dins de la variable. Aquest for fa exactament el mateix que l'anterior.

Finalment, podem utilitzar la notació funcional

```
llista.forEach(alumne->System.out.println(alumne));
```

En aquest cas a cada iteració és guarda un valor de la llista dins *alumne* i s'utilitza dins les instruccions següents.

Pila

Una pila és una seqüència amb la particularitat que els elements s'insereixen i es recuperen per el mateix costat de la llista. Només es pot inserir un element a la part superior de la pila i només es pot recuperar l'element de la part superior de la pila.

Les operacions que es permeten a una pila són les següents:

- isEmpty(): torna true si la pila és buida i false en cas contrari.
- push(x): posa l'element x a la pila
- peek(): Torna l'element de la part superior de la pila, però no l'esborra.
- pop(): Esborra l'element de la part superior de la pila, però no el torna.
- poll(): Torna i esborra l'element de la part superior de la pila.

Exemple: CtrlZ

Algunes de les classes de l'API que defineixen una pila

- Stack: Implementa totes les operacions d'una pila, però el seu rendiment no és el millor per el fet de ser sincronitzada.
- LinkedList: Llista doblement encadenada, cada element apunta al següent i a l'anterior. Inclou totes les operacions definides per a piles.
- ArrayDeque: Implementa les operacions d'una pila i una cua i ofereix millor rendiment que Stack i LinkedList.

Podeu trobar l'ajuda d'aquestes classes a l'API de Java, cercant JAVA <versió que utilitzeu> API i el nom de la classe.

(Parèntesi: genèrics)

De vegades ens pot ser útil definir una classe de manera que el tipus d'un o diversos atributs no estigui establert en la definició de la classe.

Pot ser útil per exemple, en la definició de la pila que hem fet. La versió que tenim feta només ens serveix per emmagatzemar Integer, però seria molt més aprofitable si hi poguéssim guardar altres tipus d'objectes. Tenim dues opcions:

- Definir el camp *info* com a *Object*. Això ens permetrà posar-hi qualsevol objecte, fins i tot podrem afegir objectes de diferents tipus a la mateixa Pila. En recuperar-los, però, haurem de fer el càsting cap a la classe original de l'objecte.
- Parametritzar la classe de manera que quan declarem un objecte de classe Pila especifiquem la classe dels elements que podrà contenir. Podrem crear una objecte Pila<Carta> que només admetrà objectes d'aquest tipus i un altre objecte Pila<Alumne> que només admetrà alumnes. En recuperar els objectes no es necessari fer el càsting.

La parametrització es fa utilitzant una característica de Java que es diu tipus genèrics. Per exemple, si tenim la classe *Trasto* amb la següent definició:

```
public class Trasto {
private Integer valor;
public void setValor(Integer valor){
this.valor=valor;
}
... la resta del codi
}
```

només permet utilitzar Integer. Si volem poder crear objectes d'aquesta classe que puguin treballar amb diferents classes l'hem de parametritzar:

```
public class Trasto <T>{
private T valor;
public void setValor(T valor){
this.valor=valor;
}
... la resta del codi
}
```

El tipus parametritzat es posa a la definició de la classe, entre < i >, i es pot utilitzar a dins com si fos una classe qualsevol: *valor* serà de tipus T, *setValor* té un paràmetre de tipus T,...

Si cream les següents variables:

```
Trasto <String> trastodeStrings=new Trasto<>();

Trasto <Alumne> trastodeAlumnes=new Trasto<>();
```

la variable *trastoStrings* només podrà treballar amb cadenes de text, mentre que la variable *trastoAlumnes* només podrà treballar amb *Alumne*.

```
trastoCadenes.setValor(new Alumne());  //donaria error en temps de
                                                    //compilació.
```

Una classe pot utilitzar més d'un tipus parametritzat:

```
public class Andromina <V, K>{
private V valor;
private K clau;
... la resta del codi
}
```

Coa

Una coa és una seqüència d'elements amb la particularitat que els elements s'insereixen per un costat de la llista i es recuperen per l'altre.

Les operacions que es permeten a una coa són les següents:

- isEmpty(): torna true si la coa és buida i false en cas contrari.
- add(x): posa l'element x al final de la coa
- remove(): Torna l'element de la capçalera de la coa i l'esborra.
- element(): Torna l'element de la capçalera de la coa, però no l'esborra.
- clear(): Esborra tots els elements de la coa.
- size(): Torna el nombre d'elements que hi ha a la coa.

Algunes de les classes de l'API que defineixen una coa

- LinkedList: Llista doblement encadenada, cada element apunta al següent i a l'anterior. Inclou totes les operacions definides per a coes.
- ArrayDeque: Implementa les operacions d'una pila i una coa i ofereix millor rendiment que LinkedList.

Conjunt

Un conjunt és una col·lecció d'elements únics sense cap ordre establert. És a dir, abans d'inserir un objecte al conjunt es comprova que no hi sigui amb el mètode *equals* de l'objecte i si hi és no es permet el duplicat. A més si llistam el contingut del conjunt distintes vegades "pot" variar l'ordre en el que apareixen els elements.

Les operacions que es permeten a un conjunt són les següents:

- isEmpty(): torna true si el conjunt és buid i false en cas contrari.
- add(x): posa l'element x al conjunt. Torna true si ho ha fet.
- remove(x):Esborra l'element del conjunt True si ho ha fet.
- contains(x): True si el conjunt conté l'element x.
- clear():Esborra tots els elements del conjunt.
- size(): Torna el nombre d'elements que hi ha al conjunt.

Algunes de les classes de l'API que defineixen un conjunt

- HashSet: Implementa un conjunt tal i com l'hem descrit. Afegeix algunes operacions per fer feina amb col·leccions d'elements. Els elements que conté es poden recórrer amb un for(:).
- TreeSet: Implementa un conjunt ordenat.

Recórrer els elements d'un HashSet

```
for (Preferencia pref : conjunt) {
        System.out.print(pref + "\t");
}
```

```
conjunt.forEach((pref)->{
        System.out.print(pref + "\t");
});
```

(Parèntesi: mètodes equals i hashCode)

Molt sovint les classes que implementen contenidors han de comparar dos objectes entre si. L'API defineix aquests dos mètodes per a poder comparar dos objectes:

El mètode *hashCode* torna un sencer que representa l'objecte. Es calcula a partir dels seus atributs. No és únic, però ha de garantir que mentre no canviïn els atributs tornarà el mateix valor.

El mètode *equals* torna un booleà dient si l'objecte actual, this, és igual que el que passam com a argument. Per regla general aquest mètode ha de comprovar:

- Que no sigui null.
- Que siguin de la mateixa classe.
- Que cada atribut tengui el mateix valor als dos objectes.

Els IDE ens permeten generar aquests mètodes indicant els camps que volem utilitzar.

Mapa

Un mapa relaciona claus amb valors. Permet l'accés eficient a un determinat valor a partir de la seva clau sense necessitat de recórrer tots els elements.

No pot contenir claus duplicades. Podem veure la clau del mapa com una clau primària d'una taula.

Les comparacions es fan amb el mètode *equals*.

Les operacions que es permeten a un mapa són les següents:

- isEmpty(): torna true si el conjunt és buid i false en cas contrari.
- put(clau, valor): Afegeix el mapatge clau->valor al mapa. Si la clau ja existeix, substitueix el valor que hi havia per el nou. Torna el valor anterior(null si el maptge no existia)
- get(clau): Torna el valor associat a la clau.
- remove(clau):Esborra el mapatge del mapa.
- containsKey(clau): True si el conjunt conté la clau.
- containsValue(valor): True si el conjunt conté el valor.
- clear():Esborra tots els elements del mapa.
- size(): Torna el nombre d'elements que hi ha al mapa.

Classes de l'API que implementen un mapa

- HashMap: Implementa un mapa tal i com l'hem descrit. Afegeix algunes operacions per fer feina amb col·leccions d'elements. El mètode *values* ens torna una llista amb els valors que conté el mapa i *keySet* amb les claus. Es poden recórrer amb un for(:).

- Properties: Cas especial de mapa on tant les claus com els valors són de tipus String. Es sol utilitzar per guardar dades de configuració, per exemple els paràmetres d'una connexió a base de dades. Es recomana utilitzar els mètodes setProperty i getProperty per afegir o recuperar valors al mapa.

Recórrer els valors d'un HashMap

```
for (Preferencia valor:mapa.values()) {
        System.out.print(valor + "\t");
}
```

```
mapa.values().forEach(
    pref -> System.out.print(pref+"\t")
);
```

Recórrer les claus d'un HashMap

```
for (String clau : mapa.keySet()) {
        System.out.print(clau + "\t");
}
```

```
mapa.keySet().forEach(
    clau -> System.out.print(clau+"\t")
);
```

Recórrer les entrades d'un HashMap (Objectes amb el parell clau, valor)

```
for (String entrada:mapa.entrySet()) {
    System.out.print(entrada.getKey()
        +" "+entrada.getValue()+"\t"));
}
```

```
mapa.entrySet().forEach(
  entrada ->
    System.out.print(entrada.getKey()
        +" "+entrada.getValue()+"\t"));
```

Enumeracions

Una enumeració és un tipus de dades on els seus atributs són un conjunt fixe de constants. Exemples d'enumeracions podrien ser els mesos de l'any, els punts cardinals, ...

Com que son constants, els noms dels atributs s'escriuen amb majúscules

La definició en Java es faria de la següent manera, **dins un fitxer apart amb el mateix nom que l'enumeració**:

```
public enum Dies {

DILLUNS, DIMARTS, DIMECRES, DIJOUS, DIVENDRES, DISSABTE, DIUMENGE

}
```

Es solen utilitzar enumeracions sempre que tenguem un **conjunt fixat de constants que coneixem en temps de compilació**. Utilitzar-les ens dona l'avantatge que el control dels valors es pot fer en temps de compilació i que són més fàcils d'utilitzar per al programador que no un conjunt de números sencers.

El mètode *values* torna un array amb els valors de les constants de l'enumeració.

Per exemple, si hem d'utilitzar els dies de la setmana en una aplicació ho podem fer utilitzant sencers:

```
private void doi(){
  int avui=2;
  if(avui==5)     System.out.println("A la fi!!!");
  }
```

Res m'impedeix assignar a avui el valor 2345 o -12. En canvi utilitzant l'enumeració definida abans:

```
private void doi(){
  Dies avui;
  avui=Dies.DIMARTS;
  if(avui==Dies.DIVENDRES{
          System.out.println(“A la fi!!!”);
  }
}
```

A part de que queda més clar que representa la variable i quin valor conté, és impossible posar un valor erroni perquè el compilador ho controla.

Enumeracions Avançat. Fora de temari.

Les enumeracions en el fons són classes. Podem definir atributs (han de ser constants), constructors per inicialitzar els atributs i mètodes.

De totes maneres no podem crear objectes o afegir valors a l'enumeració en temps d'execució.

L'exemple del costat defineix una enumeració amb els planetes del nostre sistema solar. Per a cada un d'ells definim dos atributs, la massa i el radi, el constructor, els getters i dos mètodes.

A l'inici del codi declara les diferents constants de l'enumeració i crida al constructors amb els valors de cada planeta.

```
public enum Planet {
    MERCURI(3.303e+23, 2.4397e6),
    VENUS(4.869e+24, 6.0518e6),
    TERRA(5.976e+24, 6.37814e6),
    MART(6.421e+23, 3.3972e6),
    JUPITER(1.9e+27, 7.1492e7),
    SATURN(5.688e+26, 6.0268e7),
    URÀ(8.686e+25, 2.5559e7),
    NEPTÚ(1.024e+26, 2.4746e7);

    private final double mass;   // in kilograms
    private final double radius; // in meters
    // universal gravitational constant  (m3 kg-1 s-2)
    public static final double G = 6.67300E-11;

    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
    private double mass() {
        return mass;
    }
    private double radius() {
        return radius;
    }
    double surfaceGravity() {
        return G * mass / (radius * radius);
    }
    double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}
```

Collections

Es tracta d'una **classe d'utilitat** que ofereix una sèrie de mètodes de classe que permeten operar sobre col·leccions: llistes, piles, ...

- Ordenació: sort per ordenar, shuffle per mesclar, reverse per girar-la al revés...
- Màxims i mínims: max torna el màxim de la llista, min el mínim.
- Cerca: binarySearch torna la posició de l'element que cercam.
- Substitucions: replaceAll canvia totes les aparicions d'un element per un altre, fill canvia tots els elements de la llista per el que li passam, swap intercanvia els elements que ocupen les posicions que li passam, ...