IMPLEMENTAREA BIBLIOTECILOR ÎN JAVA

Construirea de biblioteci abstracte (clase si pachete de clase) este o parte importanta in dezvoltarea aplicatiilor Java si nu numai. Voi incerca prin articolele acestei rubrici sa va trezesc curiozitatea privind modalitatile de abstractizarea a datelor si construirea de noi biblioteci de date abstracte. Toate aplicatiile existente in numerele anterioare ale revistei (nr. 1 - 21) folosesc caracteristici si clase existente in Java care sunt puse la dispozitie de bibliotecile standard (java.util, java.awt, etc.). Pentru inceput voi prezenta cateva lucruri de baza in realizarea unor biblioteci abstracte de date, urmand ca in articolole viitoare sa trecem la implementarea unei clase abstracte.

Metode mostenite din Clasa Object

Clasa Object declara un numar de metode care pot fi suprascrise de subclase ale ei (acesta inseamna ca in orice aplicatie putem suprascrie aceste metode). Cand implementam o clasa trebuie sa tinem cont de unele aspecte legate de obiectele instanta (cum trebuie copiate <cloned>comparate, sterse, afisate sub forma unui String). Putem suprascrie aceste metode atunci cand comportamentul lor implicit nu satisface cerintele programului. Urmatoarele metode pot fi supraincarcate: <xmp>public boolean equals(Object obj); public String toString(); public final native int hashCode(); protected native Object clone(); protected void finalize();</xmp> Obs: obiectele de tip array permit de asemenea suprascrierea acestor metode.

Trei dintre aceste metode sunt publice si pot fi suprascrise de orice instante obiect, in timp ce doua metode sunt protejate si din acest motiv trebuie declarate publice in momentul in care sunt suprascrise. Vom analiza pe rand aceste metode:

boolean equals(Object obj)

Metoda folosita pentru a compara doua obiecte (obiectul pentru care se apeleaza metoda si obiectul transmis ca parametru). Metoda implicita oferita de clasa Object returneaza true daca cele doua obiecte reprezinta de fapt acelasi obiect, folosindu-se operatorul == . Ramane in sarcina programatorului sa decida cum se compara doua obiecte ale aceleeasi clase.

Documentatia JDK defineste un set riguros de reguli ce trebuie avute in vedere atunci cand se doreste stabilirea egalitatii intre boua obiecte. Metoda equals implementeaza o relatie de echivalenta:

- Este reflexiva ;
- Este simetrica;
- Este tranzitiva;

String toString()

Metoda returneaza o reprezentare de tip String pentru obiectul care o apeleaza. Implicit returneaza un Sring sub forma:

ClassName @ 1cc7a0, adica numele clasei urmat de caracterul @ si apoi o valoare in hexa a codului hash. Pentru a genera o reprezentare mult mai utila putem supraincarca acesta metoda si returna orice String care sa ne ofere informatii despre obiect.

int hashCode()

Un hash cod este o valoare intreaga ce reprezinta intreaga valoare a unui obiect. Codurile hash sunt folosite drept chei in tabelele de dispersie asa cum este implementata clasa HashTable din pachetul java.util. Versiunea implicita a metodei va incerca sa genereze un cod pentru fiecare obiect dar se poate ca la un moment dat sa genereze valori diferite pentru un acelasi obiect. Daca se intampla acest lucru atunci trebuie sa suprascriem metoda pentru a implementa o noua functie de dispersie (hash function) care va genera codurile hash corecte.

De fiecare data cand este invocata metoda hashCode asupra aceluiasi obiect ea trebuie sa returneze in mod constant aceeasi valoare intreaga. Daca doua obiecte sunt egale conform metodei equals, atunci apeland metoda hashCode pentru fiecare din cele doua obiecte trebuie sa obtinem acceasi valoare intreaga.

Programatorii se bazeaza de obicei pe implementarea implicita a metodei hashCode decat sa implementeze o noua versiune (ceea ce poate duce la o munca destul de dificila).

object clone()

Metoda va crea o copie a obiectului. Implicit doar obiectul curent este copiat si nu si celelalte obiecte spre care acesta poate avea referinte. Valorile primitive in Java sunt intotdeauna copiate. Metoda suprascrisa trebuie declarata public. Daca un obiect nu poate fi clonat va fi aruncata exceptia : CloneNotSupportedException.

void finalize()

Acesta metoda este apelata automat de colectorul de gunoaie (garbage collector) cand un obiect nu mai este referentiat si poate fi sters din memorie. Varianta implicita nu contine nici o instructiune in corpul metodei. Colectorul de gunoaie poate rula oricand, astfel incat nu se poate determina cu exactitate cand va fi apelata metoda finalize. Putem supraincarca acesta metoda in cazul in care de exemplu anumite date trebuie salvate intr-un fisier inainte de a fi pierdute sau o conexiune pe retea trebuie inchisa.

Daca apare o eroare metoda poate folosi in declaratie si clauza throw, aruncand o exceptie de tipul Throwable. Daca acesta exceptie este aruncata atunci ea va fi prinsa de colectorul de gunoaie si ignorata, lasand programul sa-si desfasoare executia pana la final.

Cam atat cu teoria !!! :-) Sa trecem la fapte, adica sa discutam pe exemple concrete:

Presupun cunoscut modul cum se compileaza programele in Java si cum se ruleaza. Pentru incepatori recomand cartea Thinking in Java (Second Edition) de Bruce Eckel, care poate fi download-ata de pe site-ul autorului: www.bruceeckel.com). Incepem cu implementarea clasei ObjectMethods.java: <xmp>import java.util.Date; class ObjectMethods //extinde implicit clasa Object { public ObjectMethods(int a, Date b, String[]c) //constructor cu parametri { i = a; d = b; s = c; } private ObjectMethods() //constructor fara parametri {} public boolean equals(Object obj) //mostenita din Object si suprascrisa { if (this == obj) return true; if ((obj == null) || !(obj instanceof ObjectMethods)) return false; ObjectMethods tmp = (ObjectMethods)obj; if (i != tmp.i) return false; if (!d.equals(tmp.d)) return false; if (s.length != tmp.s.length) return false; /* Un sir este un obiect. Apeland == sau equals se va verifica doar daca doua referinte indica acelasi obiect; de aceea un ciclu este necesar pentru a parcurge toate elementele vectorului si ale compara. */ for(int i = 0;i<s.length i if s i equals tmp.s i return false return true/gereram aici o reprezentare o obiectului //sub forma de String public String toString() { StringBuffer sb = new StringBuffer(); sb.append("i= "+i+", "); sb.append("d= "+d+", "); sb.append("s= ");

for(int i=0;i<s.length i sb.append s i quot quot return sb.tostring public object clone/realizeaza o copie exacta o obiectului { ObjectMethods tmp = new ObjectMethods(); tmp.i = i; //nu putem clona un obiect de tip Date(), asa //ca vom crea un nou obiect initializat //cu valoarea lui d tmp.d = new Date(d.getTime()); tmp.s = new String[s.length]; for(int i = 0;i < s.length;i++) { tmp.s[i] = new String(s[i]); } return tmp; } public void finalize() { for(int i=0;i/true System.out.println(a.equals(b)); //false System.out.println(b.equals(a)); //false System.out.println(b.equals(b)); //true System.out.println(a); System.out.println(b); b = null; ObjectMethods c = (ObjectMethods)(a.clone()); System.out.println(a.equals(c));//true System.out.println(c.equals(a));//true System.out.println(a); System.out.println(c); //modific obiectul c (obiectul a //nu se modifica!!!!!)) c.modify(22); System.out.println("NEW c:"+c); System.out.println(" a:"+a); ObjectMethods m ;//= new ObjectMethods m = a; //m este referinta la a : daca //modific m se modifica si a !!!!!!! m.modify(99); System.out.println("m: "+m); System.out.println("a: "+a); System.out.println("NEW m:"+m); System.out.println("New a:"+a); } }</xmp>