Strukture podataka – ArrayList implementacija

ArrayList

U programskom jeziku Java, klasa `java.util.ArrayList` je lista koja za internu reprezentaciju podataka koristi niz. Implementacija ove klase je dostupna na linku http://hg.openjdk.java.net/jdk7/jdk7/jdk/file/tip/src/share/classes/java/util/ArrayList.java. U implementaciji ove klase ima ima detalja koje za sada možemo da zanemarimo, pa smo zbog toga implementirali pojednostavljenu verziju ove klase: `vp.liste.ArrayList`. Ta implementacija je nastala preuzimanje ilustrativnog programskog koda iz originalne implementacije `java.util.ArrayList`. Implementacija je prikazana u narednim listinzima koda.

```
public class ArrayList<E> {

	private Object[] elementData;

	private int size;

	public ArrayList() {
		this.elementData = new Object[10];
	}
```

Listing – implementacija ArrayList klase – predstava elemenata

ArrayLista je implementirana kao generička klasa da bi prilikom deklarisanja mogao da se navede tip objekata koji će se čuvati u listi. Elementi liste predstavljeni su kao niz `elementData` instanci klase `Object`, što znači da će bilo koji objekat moći da bude dodat u listu. Broj elemenata liste predstavljen je atributom `size`. Prilikom kreiranja objekta ArrayList inicijalno se napravi niz `elementData` kapaciteta 10.

```
	private void grow(int minCapacity) {
		int oldCapacity = elementData.length;
		int newCapacity = oldCapacity + (oldCapacity / 2);
		if (newCapacity - minCapacity < 0)
			newCapacity = minCapacity;
		elementData = Arrays.copyOf(elementData, newCapacity);
	}
	private void ensureCapacityInternal(int minCapacity) {
		if (minCapacity - elementData.length > 0)
			grow(minCapacity);
	}
```

Listing – implementacija ArrayList klase – rast liste

Inicijalno je napravljen niz koji ima 10 elemenata. Kada budemo hteli da dodamo 11. element u listu, niz moramo „proširiti“. To radimo tako što kapacitet novog niza za polovinu uvećamo u odnosu na kapacitet starog niza (`int newCapacity = oldCapacity + (oldCapacity / 2);`), i iskopiramo elemente iz starog niza na početak novog niza (`elementData = Arrays.copyOf(elementData, newCapacity);`). Obratite pažnju da možemo proslediti minimalni kapacitet novog niza (`minCapacity`).

Metoda `ensureCapacityInternal` vrši proveru da li je niz `elementData` dovoljno velik da primi nove vrednosti i, ako nije, poziva metodu `grow` koja će „proširiti“ niz do željene veličine.

```
	public int size() {
		return size;
	}

	public boolean isEmpty() {
		return size == 0;
	}
```

Listing – implementacija ArrayList klase – provera veličine liste

Veličinu liste ćemo preuzeti kao vrednost varijable `size`, a provera da li je lista prazna svodi se na proveru da li `size` ima vrednost `0`;

```
	public int indexOf(Object o) {
		if (o == null) {
			for (int i = 0; i < size; i++)
				if (elementData[i] == null)
					return i;
		} else {
			for (int i = 0; i < size; i++)
				if (o.equals(elementData[i]))
					return i;
		}
		return -1;
	}

	public boolean contains(Object o) {
		return indexOf(o) >= 0;
	}
```

Listing – implementacija ArrayList klase – pronalaženje objekta

Pronalaženje elementa liste realizovani je kao linearna pretraga: jedan po jedan element liste upoređujemo sa traženim objektom koristeći metodu `equals`. Ukoliko je objekat pronađen, vraćamo njegov indeks. Ukoliko ni poslednji element liste nije traženi objekat, vraćamo vrednost -1.

```
	E elementData(int index) {
		return (E) elementData[index];
	}

	public E get(int index) {
		return elementData(index);
	}
```

Listing – implementacija ArrayList klase – preuzimanje elementa na zadatoj poziciji

Preuzimanje elementa sa zadate pozicije svodi se na kastovanje elementa u tip kojim je lista parametrizovana i vraćanje te vrednosti.

```
	public E set(int index, E element) {
		E oldValue = elementData(index);
		elementData[index] = element;
		return oldValue;
	}
```

Listing – implementacija ArrayList klase – izmena

Izmena vrednosti na zadatoj poziciji svodi se na preuzimanje stare vrednosti, postavljanje nove vrednosti na tu poziciju i vraćanje stare vrednosti.

```
public boolean add(E e) {
  ensureCapacityInternal(size + 1);
  elementData[size++] = e;
  return true;
}

public void add(int index, E element) {
	ensureCapacityInternal(size + 1);
	System.arraycopy(elementData, index, elementData, index + 1, size
			- index);
	elementData[index] = element;
	size++;
}
```

Listing – implementacija ArrayList klase – dodavanje elementa

Kada dodajemo element na kraj list (`public boolean add(E e)`), prvo se obezbeđuje da niz `elementData` ima makar jedno mesto više nego što je veličina lista. Zatim se posle poslednjeg elementa doda novi element i `size` se inkrementira (uveća za 1).

Dodavanje elmenta na željenu poziciju u listu implementirano je metodom `public void add(int index, E element)`. Pri tome se prvo obezbeđuje da niz `elementData` ima makar jedno mesto više nego je trenutna veličina liste (da bi novi element mogao da „stane“). Zatim se svi elementi `elementData` niza iza pozicije na koju se dodaje novi element pomere za jedno mesto udesno (`System.arraycopy(elementData, index, elementData, index + 1, size - index);`). Na kraju se novi element postavi na željenu poziciju i veličina liste inkrementira. Vidimo da ovo nije naročito efikasna implementacija: ukoliko bismo hteli da novi element dodamo na početak liste, morali bismo da presnimimo sve element iz liste. Pored toga, obzirom da ove dve metode pozivaju `ensureCapacityInternal` metodu, možemo da zaključimo da će dodavanje novog elementa u listu povremeno uključivati i kopiranje čitavog `elementData` niza.

```
public E remove(int index) {
	E oldValue = elementData(index);
	int numMoved = size - index - 1;
	if (numMoved > 0)
		System.arraycopy(elementData, index + 1, elementData, index,
				numMoved);
	elementData[--size] = null;
	return oldValue;
}

public boolean remove(Object o) {
	if (o == null) {
		for (int index = 0; index < size; index++)
			if (elementData[index] == null) {
				remove(index);
				return true;
			}
	} else {
		for (int index = 0; index < size; index++)
			if (o.equals(elementData[index])) {
				remove(index);
				return true;
			}
```

```
                }
                return false;
        }

        public void clear() {
                for (int i = 0; i < size; i++)
                        elementData[i] = null;
                size = 0;
        }
}
```

Listing – implementacija ArrayList klase – dodavanje elementa

Brisanje sa zadate pozicije implementirano je metodom `public E remove(int index)` i svodi se na kopiranje svih elemenata sa desne strane zadate pozicije za po jedno mesto u levo i postavljanje vrednosti poslednjeg elementa na null. Naravno, ova metoda uključuje i dekrementiranje veličine liste.

Brisanje zadatog objekta iz lista (`public E remove(Object o)`) podrazumeva pronalazak pozicije tog objekta linearnom pretragom i brisanje elementa sa pronađene pozicije.

Metoda `public void clear()` obrisaće sve elemente liste (postaviti ih na null) i postaviti veličinu liste na 0.

Primer – studentska služba

Sada ćemo klasu ArrayList da iskoristimo za implementiranje jednostavnih funkcionalnosti studentske službe.

Na listingu ispod prikazana je klasa Student. Student je opisan svojim brojem indeksa, imenom prezimenom i godinom studija.

```
public class Student {

        public Student(String brojIndeksa, String ime, String prezime,
                        int godinaStudija) {
                this.brojIndeksa = brojIndeksa;
                this.ime = ime;
                this.prezime = prezime;
                this.godinaStudija = godinaStudija;
        }

        String brojIndeksa;
        String ime;
        String prezime;
        int godinaStudija;


}
```

Listing – Klasa Student

Želimo da studentska služba ima listu studenata, da možemo da upišemo novog studenta, da izbrišemo studenta za zadati broj indeksa i da studenta prevedemo u narednu godinu. Klasa `StudentskaSluzba` prikazana je listingom ispod. Da bismo implementirali ovu klasu bilo nam je potrebno da znamo da klasa ArrayList ima metodu za dodavanje novog elementa, pristupanje

elementu na zadatoj poziciji i brisanje elementa. Nije nam bilo potrebno da znano kako su ove metode implementirane kao ni koja struktura podataka je iskorišćena za predstavljanje elemenata liste. Takođe, nije nam bilo potrebno da znamo ni da klasa ArrayList ima metodu `ensureCapacityInternal` koja garantuje da će u nizu biti dovoljno mesta da se novi element doda. Čitava ova metoda je detalj implementacije od kog se možemo (i trebamo) apsrahovati prilikom korišćenja klase ArrayList – zbog toga je njen modifikator pristupa `private`. Da bismo koristili neku klasu, treba nam da znamo samo koje javne (`public`) metode i polja ima.