Περιεχόμενα

Αφαίρεση δεδομένων 5

Σε αυτό το κεφάλαιο εξετάζουμε τον πιο σημαντικό μηχανισμό αφαίρεσης: την αφαίρεση δεδομένων (data abstraction). Η αφαίρεση δεδομένων μάς επιτρέπει να "αποστασιοποιηθούμε" από τις λεπτομέρειες του τρόπου υλοποίησης των αντικειμένων δεδομένων και να εστιάσουμε στον τρόπο που συμπεριφέρονται. Αυτή η εστίαση στη συμπεριφορά των αντικειμένων αποτελεί τη βάση του αντικειμενοστρεφούς προγραμματισμού.

Η αφαίρεση δεδομένων μάς επιτρέπει να επεκτείνουμε τη γλώσσα προγραμματισμού που χρησιμοποιούμε (π.χ. Java), με νέους τύπους δεδομένων. Οι νέοι τύποι δεδομένων που θα χρειαστούν εξαρτώνται από το πεδίο εφαρμογής του προγράμματος. Για παράδειγμα, κατά την υλοποίηση ενός μεταγλωττιστή (compiler) ή ενός διερμηνευτή (interpreter) είναι χρήσιμες οι στοίβες και οι πίνακες συμβόλων, ενώ οι τραπεζικοί λογαριασμοί είναι μια φυσική αφαίρεση σε ένα τραπεζικό σύστημα. Τα πολυώνυμα προκύπτουν σε ένα σύστημα χειρισμού συμβόλων, και οι μήτρες είναι χρήσιμες στον ορισμό ενός πακέτου αριθμητικών συναρτήσεων. Σε κάθε περίπτωση, η αφαίρεση δεδομένων αποτελείται από ένα σύνολο αντικειμένων — για παράδειγμα, στοίβες ή πολυώνυμα — και ένα σύνολο πράξεων ή λειτουργιών. Για παράδειγμα, οι πράξεις με μήτρες περιλαμβάνουν πρόσθεση, πολλαπλασιασμό, κ.ο.κ., ενώ η κατάθεση και η ανάληψη είναι λειτουργίες τραπεζικών λογαριασμών.

Οι νέοι τύποι δεδομένων θα πρέπει να ενσωματώνουν αφαίρεση και μέσω παραμετροποίησης (abstraction by parameterization) και μέσω προδιαγραφής (abstraction by specification). Η αφαίρεση μέσω παραμετροποίησης μπορεί να επιτευχθεί με τον ίδιο τρόπο όπως και στις διαδικασίες — με τη χρήση παραμέτρων όποτε είναι λογικό να το κάνουμε. Για να εφαρμόσουμε αφαίρεση μέσω προδιαγραφής, κάνουμε τις λειτουργίες τμήμα του τύπου. Για να καταλάβετε γιατί είναι απαραίτητες οι λειτουργίες, αναλογιστείτε τι θα συμβεί αν θεωρήσουμε ότι ένας τύπος είναι απλώς ένα σύνολο αντικειμένων. Έτσι, το μόνο που χρειάζεται για την υλοποίηση του τύπου είναι να επιλέξετε μια αναπαράσταση για την αποθήκευση των αντικειμένων· όλα τα προγράμματα που χρησιμοποιούν τον τύπο μπορούν να υλοποιηθούν με βάση αυτή την αναπαράσταση. Αν όμως αλλάξει η

αναπαράσταση, ή ακόμη και η υλοποίησή της, όλα τα προγράμματα που χρησιμοποιούν τον τύπο θα πρέπει να αλλάξουν· δεν υπάρχει τρόπος να περιοριστούν οι επιπτώσεις της αλλαγής.

Από την άλλη, υποθέστε ότι έχουμε συμπεριλάβει και λειτουργίες στον τύπο, έτσι ώστε

αφαίρεση δεδομένων = {αντικείμενα, λειτουργίες}

και απαιτούμε από τους χρήστες να καλούν τις λειτουργίες αντί να προσπελάζουν απευθείας την αναπαράσταση. Κατόπιν, για να υλοποιήσουμε τον τύπο υλοποιούμε τις λειτουργίες με βάση την επιλεγμένη αναπαράσταση, και αν αλλάξουμε την αναπαράσταση θα πρέπει να αλλάξουμε την υλοποίηση των λειτουργιών. Ωστόσο, δε χρειάζεται να αλλάξουμε την υλοποίηση οποιουδήποτε προγράμματος χρησιμοποιεί τον τύπο, επειδή δε χρησιμοποιεί την αναπαράσταση. Τώρα έχουμε πραγματοποιήσει αφαίρεση από την αναπαράσταση των λεπτομερειών· ο κώδικας που χρησιμοποιεί τον τύπο βασίζεται μόνο στην καθορισμένη συμπεριφορά του τύπου με τις λειτουργίες του. Επομένως, έχουμε επιτύχει αφαίρεση μέσω προδιαγραφής.

Αν παρέχονται αρκετές λειτουργίες, η έλλειψη πρόσβασης στην αναπαράσταση δε θα προκαλέσει καμία δυσκολία τους χρήστες — οτιδήποτε θέλουν να κάνουν με τα αντικείμενα μπορεί να γίνει, και γίνεται αποδοτικά, με κλήσεις των λειτουργιών. Γενικά, θα υπάρχουν λειτουργίες για τη δημιουργία και την τροποποίηση των αντικειμένων και τη λήψη πληροφοριών σχετικά με τις τιμές τους. Φυσικά, οι χρήστες μπορούν να βελτιώσουν το σύνολο των λειτουργιών ορίζοντας αυτόνομες διαδικασίες, αλλά αυτές οι διαδικασίες δε θα έχουν πρόσβαση στην αναπαράσταση.

Η αφαίρεση δεδομένων μάς επιτρέπει να αναβάλουμε αποφάσεις σχετικά με τις δομές δεδομένων μέχρι να γίνουν πλήρως κατανοητές οι χρήσεις των δεδομένων. Η επιλογή των κατάλληλων δομών δεδομένων είναι κρίσιμη για τη δημιουργία ενός αποδοτικού προγράμματος. Χωρίς την αφαίρεση δεδομένων, οι δομές δεδομένων θα πρέπει να οριστούν πολύ νωρίς· πρέπει να καθοριστούν για να μπορούν να σχεδιαστούν οι υπομονάδες που θα τις χρησιμοποιήσουν. Σε αυτό το σημείο όμως, οι χρήσεις των δεδομένων συνήθως δεν έχουν κατανοηθεί πλήρως. Επομένως, η επιλεγμένη δομή μπορεί να στερείται τις απαραίτητες πληροφορίες ή να είναι οργανωμένη με μη αποδοτικό τρόπο.

Χρησιμοποιούμε την αφαίρεση δεδομένων για να αποφύγουμε τον άμεσο ορισμό της δομής: εισάγουμε τον αφηρημένο τύπο με τα αντικείμενα και τις λειτουργίες του. Οι υλοποιήσεις των υπομονάδων που θα τον χρησιμοποιήσουν μπορούν να σχεδιαστούν βάσει αυτού του αφηρημένου τύπου. Αποφάσεις που αφορούν τον τρόπο υλοποίησης του τύπου παίρνονται αργότερα, όταν θα έχουν κατανοηθεί όλες οι χρήσεις του.

Η αφαίρεση δεδομένων είναι επίσης πολύτιμη στην τροποποίηση και τη συντήρηση του προγράμματος. Σε αυτή τη φάση, οι δομές δεδομένων είναι εξαιρετικά πιθανό να αλλάξουν, είτε για τη βελτίωση της απόδοσης είτε για να συμπεριληφθούν νέες αλλαγές. Η αφαίρεση δεδομένων περιορίζει τις αλλαγές μόνο στην υλοποίηση του τύπου· καμία υπομονάδα που τον χρησιμοποιεί δε θα χρειαστεί να αλλάξει.

Σε αυτό το κεφάλαιο, περιγράφουμε πώς καθορίζεται και υλοποιείται η αφαίρεση δεδομένων στην Java. Εξετάζουμε επίσης τρόπους για την εκτίμηση της ορθότητας των προγραμμάτων που χρησιμοποιούν και υλοποιούν τύπους, και περιγράφουμε ορισμένα ζητήματα που προκύπτουν κατά το σχεδιασμό νέων τύπων.

5.1 Προδιαγραφές αφαίρεσης δεδομένων

Όπως και στην περίπτωση των διαδικασιών, η έννοια ενός τύπου δεν πρέπει να δίνεται από τις υλοποιήσεις του. Αντί γι' αυτό, η συμπεριφορά του πρέπει να ορίζεται από μια προδιαγραφή. Επειδή τα αντικείμενα του τύπου χρησιμοποιούνται μόνο με κλήσεις των λειτουργιών του, το μεγαλύτερο τμήμα των προδιαγραφών αποτελείται από εξηγήσεις για το τι κάνουν οι λειτουργίες.

Στην Java, οι νέοι τύποι ορίζονται με κλάσεις ή διασυνδέσεις. Για την ώρα, θα εξετάσουμε μόνο κλάσεις· τις διασυνδέσεις θα τις δούμε στο Κεφάλαιο 7.

Κάθε κλάση ορίζει έναν τύπο δηλώνοντας το όνομα του τύπου, ένα σύνολο *συναρτήσεων δόμησης* (constructors), και ένα σύνολο *μεθόδων παρουσίας* (instance methods) ή απλώς *μεθόδων*. Οι συναρτήσεις δόμησης χρησιμοποιούνται για την ανάθεση αρχικής τιμής σε νέα αντικείμενα, ή αλλιώς *παρουσίες* (instances), του τύπου. Από τη στιγμή που έχει δημιουργηθεί ένα αντικείμενο (και έχει πάρει αρχική τιμή από μια συνάρτηση δόμησης), οι χρήστες μπορούν να το προσπελάσουν καλώντας τις μεθόδους του.

Η μορφή μιας προδιαγραφής αφαίρεσης δεδομένων φαίνεται στην Εικόνα 5.1. Η κεφαλίδα `class dname` δηλώνει ότι ορίζεται ένας νέος τύπος με το όνομα `dname`. Η κεφαλίδα περιέχει μια δήλωση για την ορατότητα (visibility) της κλάσης· σχεδόν όλες οι κλάσεις έχουν δημόσια ορατότητα ώστε να μπορούν να χρησιμοποιηθούν από κώδικα που βρίσκεται έξω από το πακέτο που τις περιέχει.

Η προδιαγραφή αποτελείται από τρία μέρη. Το *γενικό* μέρος (overview) δίνει μια σύντομη περιγραφή της αφαίρεσης δεδομένων, όπως και μια άποψη των αφηρημένων δεδομένων με βάση "γνωστές" έννοιες. Συνήθως παρουσιάζει ένα μοντέλο των αντικειμένων· δηλαδή, περιγράφει τα αντικείμενα συναρτήσει άλλων αντικειμένων ώστε να είναι δυνατή η κατανόηση της περιγραφής από τον αναγνώστη. Για παράδειγμα, οι στοίβες μπορούν να οριστούν με βάση μαθηματικές ακολουθίες. Το γενικό μέρος δηλώνει επίσης αν τα αντικείμενα του τύπου είναι μεταλλάξιμα (mutable), οπότε μπορεί να αλλάξει η κατάστασή τους με την πάροδο του χρόνου, ή μη μεταλλάξιμα (immutable).

Το τμήμα των *συναρτήσεων δόμησης* της προδιαγραφής ορίζει τις συναρτήσεις δόμησης που δίνουν αρχική τιμή στα νέα αντικείμενα, ενώ το τμήμα των *μεθόδων* ορίζει τις μεθόδους που επιτρέπουν την προσπέλαση των αντικειμένων μετά τη δημιουργία τους.

Εικόνα 5.1 Η μορφή μιας προδιαγραφής αφαίρεσης δεδομένων

```
visibility class dname {
  //ΓΕΝΙΚΑ: Εδώ τοποθετείται μια σύντομη περιγραφή της συμπεριφοράς
  //    των αντικειμένων του τύπου

  // συναρτήσεις δόμησης
  // εδώ τοποθετούνται οι προδιαγραφές των συναρτήσεων δόμησης

  // μέθοδοι
  // εδώ τοποθετούνται οι προδιαγραφές των μεθόδων
}
```

Όλες οι συναρτήσεις δόμησης και οι μέθοδοι που εμφανίζονται στην προδιαγραφή θα είναι δημόσιες.

Οι συναρτήσεις δόμησης και οι μέθοδοι είναι διαδικασίες, και καθορίζονται με τη σημειογραφία προδιαγραφών που παρουσιάσαμε στα Κεφάλαια 3 και 4, με τις εξής διαφορές:

- Και οι μέθοδοι και οι συναρτήσεις δόμησης ανήκουν σε αντικείμενα, και όχι σε κλάσεις. Επομένως, η δεσμευμένη λέξη `static` δε θα εμφανιστεί στις κεφαλίδες των μεθόδων (επειδή αυτή η δεσμευμένη λέξη σημαίνει ότι η μέθοδος ανήκει στην κλάση και όχι στο αντικείμενο της κλάσης).
- Το αντικείμενο στο οποίο ανήκει μια μέθοδος ή συνάρτηση δόμησης είναι διαθέσιμο ως υπονοούμενο όρισμα, και μπορεί να γίνει αναφορά σε αυτό στην προδιαγραφή της μεθόδου ή της συνάρτησης δόμησης με τη δεσμευμένη λέξη `this`.

Όπως και στην περίπτωση των προδιαγραφών διαδικασιών, οι προδιαγραφές αφαιρέσεων δεδομένων έχουν τη μορφή σχολίων στον κώδικα. Όταν δημιουργείται για πρώτη φορά μια αφαίρεση δεδομένων, το μόνο που υπάρχει είναι η προδιαγραφή· απουσιάζει σχεδόν ολόκληρος ο κώδικας της κλάσης, όπως τα σώματα των μεθόδων. Αυτός ο κώδικας προστίθεται αργότερα, όταν υλοποιηθεί η αφαίρεση δεδομένων.

5.1.1 Προδιαγραφή τού τύπου `IntSet`

Η Εικόνα 5.2 δίνει μια προδιαγραφή της αφαίρεσης δεδομένων `IntSet` (σύνολο ακεραίων). Τα αντικείμενα `IntSet` είναι μη φραγμένα σύνολα ακεραίων, με λειτουργίες για τη δημιουργία ενός νέου, κενού αντικειμένου τύπου `IntSet`, τον έλεγχο αν ένας δεδομένος ακέραιος είναι στοιχείο του τύπου `IntSet`, και την προσθήκη και τη διαγραφή στοιχείων. Το γενικό τμήμα δείχνει ότι τα αντικείμενα `IntSet` είναι μεταλλάξιμα. Δείχνει επίσης ότι θα μοντελοποιήσουμε τον τύπο `IntSet` με βάση τα μαθηματικά σύνολα. Στο υπόλοιπο τμήμα της προδιαγραφής, καθορίζουμε όλες τις λειτουργίες που χρησιμοποιεί αυτό το μοντέλο.

Στην Εικόνα 5.2 χρησιμοποιείται σημειογραφία συνόλων στις προδιαγραφές των μεθόδων. Συγκεκριμένα, χρησιμοποιείται το + για την ένωση και το − για τη διαφορά συνόλων. Η Εικόνα 5.3 συνοψίζει τη σημειογραφία συνόλων που χρησιμοποιείται στο βιβλίο.

Ο τύπος `IntSet` έχει μόνο μία συνάρτηση δόμησης, η οποία δημιουργεί ένα νέο κενό σύνολο· παρατηρήστε ότι η προδιαγραφή αναφέρεται σε αυτό το νέο αντικείμενο συνόλου με τον τελεστή `this`. Επειδή η συνάρτηση δόμησης τροποποιεί πάντοτε το αντικείμενο `this` (για να του δώσει αρχική τιμή), δεν μπαίνουμε στον κόπο να δείξουμε αυτή την τροποποίηση στον όρο "τροποποιεί". Για την ακρίβεια, αυτή η τροποποίηση είναι αόρατη στους χρήστες: δεν έχουν πρόσβαση στο αντικείμενο της συνάρτησης δόμησης μέχρι να επιστρέψει αυτή η συνάρτηση και, επομένως, δεν μπορούν να παρατηρήσουν την αλλαγή κατάστασης.

Από τη στιγμή που το αντικείμενο `IntSet` υπάρχει, μπορούν να προστεθούν σε αυτό στοιχεία με την κλήση τής μεθόδου `insert` του αντικειμένου, και να διαγραφούν στοιχεία με την κλήση τής μεθόδου `remove`· και πάλι, οι προδιαγραφές αναφέρονται στο αντικείμενο με τον τελεστή `this`. Αυτές οι δύο μέθοδοι είναι *μεταλλάκτες* (mutators) αφού

Εικόνα 5.2 Προδιαγραφή της αφαίρεσης δεδομένων `IntSet`

```
public class IntSet {
  //ΓΕΝΙΚΑ: Τα αντικείμενα τύπου IntSet είναι μεταλλάξιμα, μη δεσμευμένα σύνολα
ακεραίων.
  // Ένα τυπικό αντικείμενο IntSet είναι {x1, . . ., xn}.

  // συναρτήσεις δόμησης
  public IntSet ( )
     //ΑΠΟΤΕΛΕΣΜΑ: Δίνει αρχική τιμή στο αντικείμενο this ώστε να είναι κενό.

  // μέθοδοι
  public void insert (int x)
     //ΤΡΟΠΟΠΟΙΕΙ: το αντικείμενο this
     //ΑΠΟΤΕΛΕΣΜΑ: Προσθέτει το x στα στοιχεία του this,
     //   δηλαδή, this_post = this + { x }.

  public void remove (int x)
     //ΤΡΟΠΟΠΟΙΕΙ: το αντικείμενο this
     //ΑΠΟΤΕΛΕΣΜΑ: Διαγράφει το x από το αντικείμενο this,
     //   δηλαδή, this_post = this - { x }

  public boolean isIn (int x)
     //ΑΠΟΤΕΛΕΣΜΑ: Αν το x περιέχεται στο αντικείμενο this
     //   επιστρέφει true αλλιώς επιστρέφει false.

  public int size ( )
     //ΑΠΟΤΕΛΕΣΜΑ: Επιστρέφει την πληθικότητα του αντικειμένου this.

  public int choose ( ) throws EmptyException
     //ΑΠΟΤΕΛΕΣΜΑ: Αν το αντικείμενο this είναι κενό,
     //   παράγει την εξαίρεση EmptyException, αλλιώς
     //   επιστρέφει ένα τυχαίο στοιχείο του αντικειμένου this.
}
```

τροποποιούν την κατάσταση του αντικειμένου τους· οι προδιαγραφές κάνουν σαφές ότι πρόκειται για μεταλλάκτες επειδή περιέχουν έναν όρο "τροποποιεί" που δηλώνει ότι τροποποιείται το αντικείμενο `this`. Παρατηρήστε ότι οι προδιαγραφές των `insert` και `remove` χρησιμοποιούν το συμβολισμό `this_post` για να δείξουν την τιμή του αντικειμένου `this` μετά την επιστροφή της λειτουργίας. Ένα όνομα ορίσματος εισόδου χωρίς το προσδιοριστικό `post` (μετά) σημαίνει πάντοτε την τιμή κατά την κλήση της λειτουργίας.

Οι υπόλοιπες μέθοδοι είναι *παρατηρητές* (observers): επιστρέφουν πληροφορίες για την κατάσταση του αντικειμένου τους χωρίς να αλλάζουν την κατάσταση. Οι παρατηρητές δε διαθέτουν όρο "τροποποιεί". (Για την ακρίβεια, ένας παρατηρητής δε διαθέτει όρο "τροποποιεί" που να δηλώνει ότι τροποποιείται το αντικείμενο `this`, ή κάποιο όρισμα αντικειμένου του τύπου του· συνήθως όμως οι παρατηρητές δεν τροποποιούν τίποτε.)

Εικόνα 5.3 Σημειογραφία συνόλων

Ένα σύνολο δηλώνεται ως {*x*1, …., *xn*}. Τα *xi* είναι τα στοιχεία του συνόλου. Σε ένα σύνολο δεν υπάρχουν διπλότυπα στοιχεία.

ένωση συνόλων: *t* = *s*1 + *s*2 είναι το σύνολο που περιέχει όλα τα στοιχεία του συνόλου *s*1 και όλα τα στοιχεία του συνόλου *s*2. Αν το *s*1 και το *s*2 περιέχουν ένα κοινό στοιχείο, στο σύνολο *t* θα υπάρχει μόνο μία εμφάνιση αυτού του στοιχείου.

διαφορά συνόλων: *t* = *s*1 – *s*2 είναι το σύνολο που περιέχει όλα τα στοιχεία του *s*1 τα οποία δεν είναι στοιχεία του *s*2.

τομή συνόλων: *t* = *s*1 & *s*2 είναι το σύνολο που περιέχει όλα τα στοιχεία τα οποία είναι μέλη και του *s*1 και του *s*2.

πληθικότητα: το σύμβολο |*s*| αντιπροσωπεύει το μέγεθος του συνόλου *s*.

μέλος συνόλου: η δήλωση ότι το *x* ανήκει στο *s* είναι αληθής όταν το *x* είναι στοιχείο του *s*.

σχηματισμός συνόλου: *t* = {*x* | *p*(*x*)} είναι το σύνολο όλων των στοιχείων *x* για τα οποία η *p*(*x*) είναι αληθής.

Η μέθοδος `choose` επιστρέφει ένα τυχαίο στοιχείο του συνόλου `IntSet`· έτσι, είναι υποπροσδιορισμένη (underdetermined). Αν το σύνολο είναι κενό, παράγει μια εξαίρεση. Αυτή η εξαίρεση είναι ανέλεγκτη (unchecked) αφού οι χρήστες μπορούν να καλέσουν τη μέθοδο `size` πριν καλέσουν την `choose`, για να εξασφαλίσουν εύκολα και γρήγορα ότι το σύνολο δεν είναι κενό.

Παρατηρήστε ότι η μέθοδος `insert` δεν παράγει κάποια εξαίρεση στην περίπτωση που ο ακέραιος υπάρχει ήδη στο σύνολο και, αντίστοιχα, η μέθοδος `remove` δεν παράγει εξαίρεση αν ο ακέραιος δεν περιέχεται στο σύνολο. Αυτές οι αποφάσεις βασίζονται στις παραδοχές που έχουν γίνει για τον τρόπο που θα χρησιμοποιηθούν τα σύνολα. Αναμένουμε ότι οι χρήστες θα προσθέτουν και θα διαγράφουν στοιχεία τού συνόλου χωρίς να ενδιαφέρονται αν αυτά περιέχονται ήδη στο σύνολο. Έτσι, οι μέθοδοι δεν παράγουν εξαιρέσεις. Αν αναμέναμε ένα διαφορετικό τρόπο χρήσης, θα μπορούσαμε να αλλάξουμε τις προδιαγραφές και τις κεφαλίδες αυτών των μεθόδων (ώστε να παράγουν κάποια εξαίρεση), ή θα μπορούσαμε να έχουμε επιπλέον μεθόδους που να παράγουν εξαιρέσεις (π.χ, `insertNonDup` και `removeIfIn`), ώστε οι χρήστες να μπορούν να επιλέξουν τη μέθοδο που ταιριάζει καλύτερα στις ανάγκες τους.

Στην προδιαγραφή τού συνόλου `IntSet`, βασιζόμαστε στο ότι ο αναγνώστης γνωρίζει τι είναι τα μαθηματικά σύνολα· διαφορετικά, η προδιαγραφή δεν είναι κατανοητή. Γενικά, αυτή η εξάρτηση από ανεπίσημες περιγραφές είναι και η αδυναμία των ανεπίσημων προδιαγραφών. Ίσως είναι λογικό να αναμένουμε ότι ο αναγνώστης γνωρίζει και κατανοεί μια σειρά μαθηματικών εννοιών όπως σύνολα, ακολουθίες, και ακέραιοι αριθμοί. Ωστόσο, δεν μπορούν να περιγραφούν κατάλληλα όλοι οι τύποι με τη βοήθεια παρόμοιων εννοιών. Αν οι έννοιες είναι ακατάλληλες, πρέπει να περιγράψουμε τον τύπο όσο το δυνατόν καλύ-

τερα, χρησιμοποιώντας ακόμη και εικόνες· φυσικά, πάντοτε υπάρχει ο κίνδυνος ο αναγνώστης να μην καταλάβει την περιγραφή ή να την ερμηνεύσει με διαφορετικό τρόπο. Στο Κεφάλαιο 9 θα δούμε τεχνικές για τη γραφή κατανοητών προδιαγραφών.

Σημειώστε ότι η προδιαγραφή παίρνει τη μορφή μιας προκαταρκτικής εκδοχής της κλάσης. Αυτός ο κώδικας θα μπορούσε να μεταγλωττιστεί αν οι μέθοδοι και οι συναρτήσεις δόμησης είχαν κενά σώματα (εκτός από τις μεθόδους που επιστρέφουν αποτελέσματα, οι οποίες θα χρειάζονταν την εντολή `return` με τον κατάλληλο τύπο). Αυτό θα σας επιτρέψει να μεταγλωττίσετε κώδικα που χρησιμοποιεί την αφαίρεση, οπότε θα μπορέσετε να απαλλαγείτε από τα σφάλματα που θα εντοπίσει ο μεταγλωττιστής, όπως τα σφάλματα τύπων. Ωστόσο, ίσως να μην είστε σε θέση να εκτελέσετε τον κώδικα που χρησιμοποιεί την αφαίρεση μέχρι να υλοποιηθεί ο νέος τύπος.

5.1.2 Η αφαίρεση πολυωνύμου `Poly`

Η Εικόνα 5.4 δίνει ένα δεύτερο παράδειγμα μιας προδιαγραφής αφαίρεσης δεδομένων. Τα αντικείμενα `Poly` είναι πολυώνυμα με ακέραιους συντελεστές. Σε αντίθεση με τα αντικείμενα `IntSet`, τα αντικείμενα `Poly` είναι μη μεταλλάξιμα (immutable): από τη στιγμή που θα δημιουργηθεί ένα αντικείμενο `Poly` (και θα πάρει αρχική τιμή με μια συνάρτηση δόμησης), δεν μπορεί να τροποποιηθεί. Υπάρχουν λειτουργίες για τη δημιουργία ενός αντικειμένου `Poly` με έναν όρο, και για την πρόσθεση, την αφαίρεση, και τον πολλαπλασιασμό αντικειμένων `Poly`.

Ο τύπος `Poly` έχει δύο συναρτήσεις δόμησης· μία για τη δημιουργία του μηδενικού πολυωνύμου και μία για τη δημιουργία ενός τυχαίου μονωνύμου. Γενικά, ένας τύπος μπορεί να έχει πολλές συναρτήσεις δόμησης. Όλες οι συναρτήσεις δόμησης έχουν το ίδιο όνομα — το όνομα του τύπου — και, επομένως, αν υπάρχουν περισσότερες από μία συναρτήσεις δόμησης, το όνομα αυτό *υπερφορτώνεται* (overload).

Η Java επιτρέπει επίσης και την υπερφόρτωση των ονομάτων μεθόδων, και απαιτεί οι υπερφορτωμένοι ορισμοί να διαφέρουν μεταξύ τους στο πλήθος ή τον τύπο των ορισμάτων τους· διαφορετικά, θα συμβεί σφάλμα χρόνου μεταγλώττισης. Οι δύο ορισμοί τής συνάρτησης δόμησης του τύπου `Poly` είναι έγκυροι, αφού η μια δεν έχει ορίσματα και η άλλη έχει δύο ορίσματα.

Ο τύπος `Poly` δεν έχει μεθόδους μετάλλαξης (mutator methods): καμία μέθοδος δε διαθέτει όρο "τροποποιεί". Αυτό φυσικά είναι που περιμένουμε να δούμε για μια μη μεταλλάξιμη αφαίρεση δεδομένων. Επιπλέον, οι προδιαγραφές των μεθόδων δε χρησιμοποιούν το συμβολισμό *post* που χρησιμοποιήθηκε στην προδιαγραφή του τύπου `IntSet`. Ο συμβολισμός αυτός δεν είναι απαραίτητος για μη μεταλλάξιμες αφαιρέσεις: αφού η κατάσταση ενός αντικειμένου δεν αλλάζει, οι καταστάσεις του αντικειμένου πριν και μετά είναι ίδιες.

Στον ορισμό του τύπου `Poly`, πρέπει να αποφασίσουμε αν η εξαίρεση `NegativeExponentException` θα είναι ελεγμένη ή ανέλεγκτη. Επειδή φαίνεται πιθανό οι χρήστες να αποφύγουν κλήσεις με αρνητικούς εκθέτες, είναι σωστό να κάνουμε την εξαίρεση ανέλεγκτη.

Εικόνα 5.4 Προδιαγραφή της αφαίρεσης δεδομένων `Poly`

```
public class Poly {
  //ΓΕΝΙΚΑ: Τα αντικείμενα Poly είναι μη μεταλλάξιμα πολυώνυμα με ακέραιους
  //συντελεστές.
  //  Ένα τυπικό πολυώνυμο είναι το c0 + c1x + . . .

  // συναρτήσεις δόμησης
  public Poly ( )
     //ΑΠΟΤΕΛΕΣΜΑ: Δίνει ως αρχική τιμή στο this το μηδενικό πολυώνυμο.

  public Poly (int c, int n) throws NegativeExponentException
     //ΑΠΟΤΕΛΕΣΜΑ: Αν n < 0 παράγει την εξαίρεση NegativeExponentException
     //  αλλιώς δίνει ως αρχική τιμή στο αντικείμενο this το αντικείμενο Poly cx^n.

  // μέθοδοι
  public int degree ( )
     //ΑΠΟΤΕΛΕΣΜΑ: Επιστρέφει το βαθμό του αντικειμένου this, δηλαδή,
     //το μεγαλύτερο εκθέτη με συντελεστή διάφορο του μηδενός Επιστρέφει 0 αν το this
     //  είναιτο μηδενικό Poly.

  public int coeff (int d)
     //ΑΠΟΤΕΛΕΣΜΑ: Επιστρέφει το συντελεστή του όρου του αντικειμένου this
     //  του οποίου ο εκθέτης είναι d.

  public Poly add (Poly q) throws NullPointerException
     //ΑΠΟΤΕΛΕΣΜΑ: Αν το q είναι null παράγει την εξαίρεση NullPointerException
     //  αλλιώς επιστρέφει το Poly this + q.

  public Poly mul (Poly q) throws NullPointerException
     //ΑΠΟΤΕΛΕΣΜΑ: Αν q είναι null παράγει την εξαίρεση NullPointerException
     //  αλλιώς επιστρέφει το Poly this * q.

  public Poly sub (Poly q) throws NullPointerException
     //ΑΠΟΤΕΛΕΣΜΑ: Αν q είναι null παράγει την εξαίρεση NullPointerException
     //  αλλιώς επιστρέφει το Poly this - q.

  public Poly minus ( )
     //ΑΠΟΤΕΛΕΣΜΑ: // Επιστρέφει το Poly - this.
}
```

5.2 Χρήση αφαιρέσεων δεδομένων

Η Εικόνα 5.5 δίνει παραδείγματα διαδικασιών που χρησιμοποιούν αφαιρέσεις δεδομένων. (Οι κλάσεις των διαδικασιών δε φαίνονται στην εικόνα.) Η μέθοδος `diff` επιστρέφει ένα νέο αντικείμενο πολυωνύμου `Poly`, αποτέλεσμα παραγώγισης του ορίσματός του που είναι τύπου `Poly`. Η ρουτίνα `getElements` επιστρέφει ένα αντικείμενο `IntSet` που περιέχει τους ακεραίους του πίνακα `a` του ορίσματος· στο σύνολο που επιστρέφεται δεν υπάρχουν διπλότυπα στοιχεία (αφού τα σύνολα δεν περιέχουν διπλότυπα) ακόμη και αν υπάρχουν διπλότυπα μεταξύ των στοιχείων του πίνακα `a`.

Αυτές οι ρουτίνες έχουν γραφεί με βάση τις προδιαγραφές των χρησιμοποιούμενων αφαιρέσεων, και μπορούν να χρησιμοποιήσουν μόνον ό,τι περιγράφεται στις προδιαγραφές. Δεν είναι σε θέση να προσπελάσουν τις λεπτομέρειες της υλοποίησης των αφηρημένων αντικειμένων καθώς, όπως θα δούμε, αυτή η πρόσβαση περιορίζεται στις υλοποιήσεις των συναρτήσεων δόμησης και των μεθόδων του αντικειμένου. Μπορούν να χρησιμοποιήσουν μεθόδους για να προσπελάσουν την κατάσταση του αντικειμένου και να την τροποποιήσουν (αν το αντικείμενο είναι μεταλλάξιμο) και μπορούν να χρησιμοποιήσουν συναρτήσεις δόμησης για την ανάθεση αρχικών τιμών σε νέα αντικείμενα.

Εικόνα 5.5 Χρήση αφηρημένων τύπων δεδομένων

```
public static Poly diff (Poly p) throws NullPointerException {
  //ΑΠΟΤΕΛΕΣΜΑ: Αν p είναι null παράγει την εξαίρεση NullPointerException
  // αλλιώς επιστρέφει το αντικείμενο Poly που λαμβάνεται με παραγώγιση του p.
  Poly q = new Poly ( );
  for (int i = 1; i <= p.degree( ); i++)
     q = q.add(new Poly(p.coeff(i)*i, i - 1));
  return q;
}

public static IntSet getElements (int[ ] a)
     throws NullPointerException {
  //ΑΠΟΤΕΛΕΣΜΑ: Αν a είναι null παράγει την εξαίρεση NullPointerException
  //   αλλιώς επιστρέφει ένα σύνολο που περιέχει μία καταχώριση
  //   για κάθε ξεχωριστό στοιχείο του a.
  IntSet s = new IntSet( );
  for (int i = 0; i < a.length; i++) s.insert(a[i]);
  return s;
}
```

5.3 Υλοποίηση αφαιρέσεων δεδομένων

Μια κλάση ορίζει ένα νέο τύπο και παρέχει μια υλοποίησή του. Η προδιαγραφή αποτελεί τον ορισμό του τύπου. Το υπόλοιπο της κλάσης παρέχει την υλοποίηση.

Για να υλοποιήσουμε μια αφαίρεση δεδομένων, επιλέγουμε μια *αναπαράσταση* (representation) για τα αντικείμενά της και μετά υλοποιούμε τις συναρτήσεις δόμησης για την ανάθεση αρχικής τιμής στην αναπαράσταση με τον κατάλληλο τρόπο, και τις μεθόδους για την κατάλληλη χρήση και τροποποίηση της αναπαράστασης. Η επιλεγμένη αναπαράσταση πρέπει να επιτρέπει όλες τις λειτουργίες που θα υλοποιηθούν, με ένα λογικά απλό και αποδοτικό τρόπο. Επιπλέον, αν κάποιες από τις λειτουργίες πρέπει να εκτελούνται γρήγορα, η αναπαράσταση πρέπει να παρέχει αυτή τη δυνατότητα. Μια αναπαράσταση που είναι γρήγορη για ορισμένες λειτουργίες συχνά θα είναι βραδύτερη για άλλες. Θα μπορούσαμε επομένως να έχουμε πολλές υλοποιήσεις του ίδιου τύπου· στο Κεφάλαιο 7 θα δούμε πώς μπορούμε να το καταφέρουμε αυτό.

Για παράδειγμα, μια πιθανή αναπαράσταση ενός αντικειμένου `IntSet` είναι ένα διάνυσμα, όπου κάθε ακέραιος του συνόλου `IntSet` εμφανίζεται ως στοιχείο του διανύσματος. Θα μπορούσαμε να επιτρέψουμε σε κάθε στοιχείο του συνόλου να υπάρχει μόνο μία φορά στο διάνυσμα ή να υπάρχει πολλές φορές. Η δεύτερη επιλογή κάνει την υλοποίηση της μεθόδου `insert` να εκτελείται πιο γρήγορα, αλλά καθυστερεί την εκτέλεση των `remove` και `isIn`. Επειδή το πιθανότερο είναι να καλείται πιο συχνά η `isIn`, θα διαλέξουμε την πρώτη δυνατότητα και, επομένως, δε θα υπάρχουν διπλότυπα στοιχεία στο διάνυσμα.

5.3.1 Υλοποίηση αφαιρέσεων δεδομένων στην Java

Μια αναπαράσταση συνήθως διαθέτει μια σειρά από συστατικά· στην Java, καθένα από αυτά είναι μια *μεταβλητή παρουσίας* (instance variable) της κλάσης που υλοποιεί την αφαίρεση δεδομένων. Οι υλοποιήσεις των συναρτήσεων δόμησης και των μεθόδων προσπελάζουν και χειρίζονται τις μεταβλητές παρουσιών.

Έτσι, από την άποψη της υλοποίησης τα αντικείμενα έχουν και μεθόδους και μεταβλητές παρουσιών. Για την υποστήριξη της αφαίρεσης όμως, είναι σημαντικό να περιοριστεί η πρόσβαση των μεταβλητών παρουσιών στην υλοποίηση των μεθόδων και των συναρτήσεων δόμησης· αυτό σας επιτρέπει, για παράδειγμα, να αλλάξετε την υλοποίηση ενός αφηρημένου τύπου χωρίς να επηρεάσετε τον κώδικα που χρησιμοποιεί τον τύπο. Επομένως, οι μεταβλητές παρουσιών δεν πρέπει να είναι ορατές στους χρήστες· ο κώδικας που χρησιμοποιεί τα αντικείμενα μπορεί να αναφέρεται μόνο στις μεθόδους τους.

Οι μεταβλητές παρουσιών δε γίνονται ορατές στους χρήστες αν τις δηλώσουμε ως ιδιωτικές (private). Η Java επιτρέπει στις μεταβλητές παρουσιών να έχουν και άλλη ορατότητα εκτός από την ιδιωτική. Γενικά δεν είναι καλή ιδέα να έχετε δημόσιες μεταβλητές παρουσιών· αυτό το θέμα θα το δούμε πιο αναλυτικά στις Ενότητες 5.6.2 και 5.9. Μοναδική εξαίρεση αυτού του κανόνα είναι ο ορισμός τύπων εγγραφών (record types)· οι τύποι εγγραφών εξετάζονται στην Ενότητα 5.3.4.

Οι δηλώσεις των μεταβλητών παρουσιών δεν έχουν το προσδιοριστικό `static`. Αυτές οι μεταβλητές ανήκουν σε αντικείμενα· υπάρχει ξεχωριστό σύνολό τους για κάθε αντικείμενο. Επίσης, είναι δυνατή η δήλωση στατικών μεταβλητών στο εσωτερικό μιας

κλάσης. Αυτές οι μεταβλητές ανήκουν στην ίδια την κλάση και όχι σε συγκεκριμένα αντικείμενα, όπως ακριβώς ανήκουν στην κλάση οι στατικές μέθοδοι. Οι στατικές μεταβλητές δε χρησιμοποιούνται πολύ συχνά στην υλοποίηση αφαιρέσεων δεδομένων· μερικά παραδείγματα που αφορούν τη χρήση τους θα δούμε στο Κεφάλαιο 15.

5.3.2 Υλοποίηση του τύπου `IntSet`

Αυτή η ενότητα δίνει ένα πρώτο παράδειγμα υλοποίησης — για την αφαίρεση δεδομένων `IntSet`. Η υλοποίηση παρουσιάζεται στην Εικόνα 5.6.

Το πρώτο θέμα που πρέπει να παρατηρήσουμε εδώ είναι ο ορισμός της αναπαράστασης του τύπου `IntSet`, πριν από τις υλοποιήσεις των συναρτήσεων δόμησης και των μεθόδων. Σε αυτή την περίπτωση, η αναπαράσταση αποτελείται από μία μόνο μεταβλητή παρουσίας. Επειδή αυτή η μεταβλητή έχει ιδιωτική (private) ορατότητα, μπορεί να προσπελαστεί μόνον από κώδικα που βρίσκεται στο εσωτερικό της κλάσης.

Οι συναρτήσεις δόμησης και οι μέθοδοι ανήκουν σε ένα συγκεκριμένο αντικείμενο του τύπου τους. Το αντικείμενο μεταβιβάζεται ως ένα επιπλέον, υπονοούμενο όρισμα στις συναρτήσεις δόμησης και στις μεθόδους, και αυτές μπορούν να αναφερθούν σε αυτό με τη χρήση της δεσμευμένης λέξης `this`. Για παράδειγμα, η μεταβλητή παρουσίας `els` μπορεί να προσπελαστεί με τη χρήση της μορφή `this.els`. (Ο κώδικας δεν μπορεί να αναθέσει τιμή στο `this`.) Το πρόθεμα δεν είναι απαραίτητο: ο κώδικας μπορεί να αναφερθεί σε μεθόδους και μεταβλητές παρουσίας του δικού του τύπου χρησιμοποιώντας απλώς το όνομά τους. Έτσι, στις μεθόδους και τις συναρτήσεις δόμησης της εικόνας, το αντικείμενο `els` αναφέρεται στη μεταβλητή παρουσίας `els` του `this`.

Η υλοποίηση του τύπου `IntSet` είναι απλή. Η συνάρτηση δόμησης δίνει αρχική τιμή στο αντικείμενό της δημιουργώντας το διάνυσμα που θα περιέχει τα στοιχεία και αναθέτοντάς το στο `els`· επειδή το διάνυσμα είναι κενό, δε χρειάζεται να γίνει τίποτε άλλο. Οι μέθοδοι `insert`, `remove`, και `isIn` χρησιμοποιούν την ιδιωτική μέθοδο `getIndex` για να προσδιορίσουν αν το στοιχείο που μας ενδιαφέρει περιέχεται ήδη στο σύνολο. Η πραγματοποίηση αυτού του ελέγχου επιτρέπει στην `insert` να διατηρήσει τη συνθήκη μη ύπαρξης διπλότυπων στοιχείων. Αυτή η συνθήκη βασίζεται στη `size` (επειδή διαφορετικά το μέγεθος του διανύσματος δε θα ήταν ίδιο με το μέγεθος του συνόλου) και στη `remove` (αφού αλλιώς θα μπορούσαν να υπάρχουν και άλλες εμφανίσεις του στοιχείου που θα έπρεπε να διαγραφούν).

Προσέξτε ότι η `getIndex` έχει ιδιωτική (private) ορατότητα· επομένως, δεν μπορεί να κληθεί έξω από την κλάση. Ο σχεδιασμός εκμεταλλεύεται αυτό το γεγονός κάνοντας την `getIndex` να επιστρέφει -1 όταν το στοιχείο δεν περιέχεται στο διάνυσμα, αντί να παραγάγει μια εξαίρεση. Όπως αναφέραμε στο Κεφάλαιο 4, αυτή η προσέγγιση είναι ικανοποιητική εδώ επειδή η `getIndex` χρησιμοποιείται μόνο στο εσωτερικό της κλάσης.

Επειδή τα διανύσματα δεν μπορούν να περιέχουν αριθμούς `int`, οι μέθοδοι χρησιμοποιούν αντικείμενα `Integer` για τα στοιχεία του συνόλου. Αυτή η προσέγγιση δεν είναι και πολύ κομψή. Μια εναλλακτική λύση είναι η χρήση πινάκων `int`· και αυτή όμως έχει τις δικές της δυσκολίες, αφού η υλοποίηση του τύπου `IntSet` θα έπρεπε να χρησιμοποιεί μεγαλύτερους πίνακες καθώς μεγαλώνει το σύνολο. Η υλοποίηση του τύπου `Vector` λύνει αυτό το πρόβλημα με έναν αποδοτικό τρόπο.

Εικόνα 5.6 Υλοποίηση του τύπου `IntSet`

```
public class IntSet {
   //ΓΕΝΙΚΑ: Τα ανικείμενα IntSet είναι μη φραγμένα, μεταλλάξιμα σύνολα ακεραίων.
   private Vector els; // η αναπαράσταση

   //συναρτήσεις δόμησης
   public IntSet ( ) {
      //ΑΠΟΤΕΛΕΣΜΑ: Δίνει ως αρχική τιμή στο αντικείμενο this το κενό.
      els = new Vector( ); }

   // μέθοδοι
   public void insert (int x) {
      //ΤΡΟΠΟΠΟΙΕΙ: το αντικείμενο this
      //ΑΠΟΤΕΛΕΣΜΑ: Προσθέτει το x στα στοιχεία του αντικειμένου this.
      Integer y = new Integer(x);
      if (getIndex(y) < 0) els.add(y); }

   public void remove (int x) {
      //ΤΡΟΠΟΠΟΙΕΙ: το αντικείμενο this
      //ΑΠΟΤΕΛΕΣΜΑ: Διαγράφει το x από το αντικείμενο this.
      int i = getIndex(new Integer(x));
      if (i < 0) return;
      els.set(i, els.lastElement( ));
      els.remove(els.size( ) - 1); }

   public boolean isIn (int x) {
      //ΑΠΟΤΕΛΕΣΜΑ: Επιστρέφει true αν το x περιέχεται στο this
      //διαφορετικά επιστρέφει false.
      return getIndex(new Integer(x)) >= 0; }

   private int getIndex (Integer x) {
      //ΑΠΟΤΕΛΕΣΜΑ: Αν το x περιέχεται στο this επιστρέφει τον αριθμοδείκτη
      //της θέσης του x, αλλιώς επιστρέφει -1.
      for (int i = 0; i < els.size( ); i++)
         if (x.equals(els.get(i))) return i;
      return -1; }

   public int size ( ) {
      //ΑΠΟΤΕΛΕΣΜΑ: Επιστρέφει την πληθικότητα του αντικειμένου this.
      return els.size( ); }

   public int choose ( ) throws EmptyException {
      //ΑΠΟΤΕΛΕΣΜΑ: Αν το this είναι κενό, παράγει την εξαίρεση EmptyException
      //   αλλιώς επιστρέφει ένα τυχαίο σοιχείο του αντικειμένου this.
         if (els.size( ) == 0) throw new
```

```
        EmptyException("IntSet.choose");
        return els.lastElement( ); }
}
```

Η `getIndex` χρησιμοποιεί τη μέθοδο `equals` για να ελέγξει αν ένα στοιχείο είναι μέλος τού συνόλου. Αυτός ο έλεγχος είναι σωστός επειδή η `equals` για αντικείμενα `Integer` επιστρέφει αληθή τιμή μόνον όταν τα δύο αντικείμενα που συγκρίνονται είναι και τα δύο τύπου `Integer` και περιέχουν την ίδια ακέραια τιμή.

5.3.3 Υλοποίηση του τύπου `Poly`

Τώρα θα εξετάσουμε την υλοποίηση της αφαίρεσης δεδομένων `Poly`. Σε αντίθεση με τα αντικείμενα `IntSet`, τα αντικείμενα `Poly` είναι μη μεταλλάξιμα και, επομένως, το μέγεθός τους δεν αλλάζει με την πάροδο του χρόνου. Άρα, μπορούμε να αναπαραστήσουμε τον τύπο `Poly` με έναν πίνακα αντί για διάνυσμα. Το $i^{στό}$ στοιχείο του πίνακα θα περιέχει το συντελεστή του $i^{στού}$ εκθέτη· αυτή η αναπαράσταση είναι λογική μόνον αν το αντικείμενο `Poly` είναι πυκνό πολυώνυμο. Το μηδενικό πολυώνυμο `Poly` μπορεί να αναπαρασταθεί είτε ως ένας κενός πίνακας είτε ως πίνακας ενός στοιχείου που περιέχει μηδέν· θα χρησιμοποιήσουμε την τελευταία προσέγγιση. Επιπλέον, θα έχουμε μια μεταβλητή παρουσίας που θα παρακολουθεί το βαθμό του αντικειμένου `Poly` αφού κάτι τέτοιο είναι βολικό.

Οι Εικόνες 5.7 και 5.8 παρουσιάζουν την υλοποίηση του τύπου `Poly`. Το κύριο ζήτημα που πρέπει να παρατηρήσουμε εδώ είναι ότι αρκετές μέθοδοι (π.χ., οι `add` και `mul`) χρησιμοποιούν μεταβλητές παρουσιών και άλλων αντικειμένων `Poly` εκτός από του δικού τους. Ο κώδικας μιας μεθόδου επιτρέπεται να προσπελάσει ιδιωτικές πληροφορίες άλλων αντικειμένων της κλάσης του, καθώς και ιδιωτικές πληροφορίες του δικού του αντικειμένου.

Παρατηρήστε πώς υλοποιούνται οι μέθοδοι `sub` και `mul` συναρτήσει άλλων μεθόδων `Poly`. Ένα άλλο θέμα είναι η χρήση της συνάρτησης δόμησης του αντικειμένου `Poly` στις υλοποιήσεις των `add`, `mul`, και `minus`. Όλες αυτές οι μέθοδοι στην πραγματικότητα δίνουν οι ίδιες αρχική τιμή στο νέο αντικείμενο `Poly`· αυτό επιτρέπεται αφού το νέο αντικείμενο `Poly` είναι απλώς ένα ακόμη αντικείμενο της κλάσης, το οποίο μπορεί να προσπελαστεί μέσα στη μέθοδο. Αυτές οι μέθοδοι δημιουργούν ένα νέο αντικείμενο `Poly` χρησιμοποιώντας την ιδιωτική συνάρτηση δόμησης (η οποία δεν μπορεί να κληθεί από χρήστες) για να πάρουν έναν πίνακα με το σωστό μέγεθος. Στην περίπτωση της `mul`, βασιζόμαστε στο γεγονός ότι η συνάρτηση δόμησης του πίνακα δίνει αρχική τιμή μηδέν σε όλα τα στοιχεία `int` του πίνακα. Επίσης, παρατηρήστε τη φροντίδα μας να εξασφαλίσουμε ότι το νέο αντικείμενο `Poly` θα έχει το σωστό μέγεθος. Αυτό απαιτεί έναν πρώτο υπολογισμό στη μέθοδο `add` για το χειρισμό των μηδενικών στο τέλος.

5.3.4 Εγγραφές

Ας υποθέσουμε ότι τα πολυώνυμα θα είναι αραιά (sparse) αντί για πυκνά (dense). Σε αυτή την περίπτωση, η προηγούμενη υλοποίηση δεν είναι κατάλληλη, αφού ο πίνακας είναι πιθανό να είναι μεγάλος και γεμάτος μηδενικά. Αντί γι' αυτό, θα θέλαμε να αποθηκεύουμε πληροφορίες μόνο για τους μη μηδενικούς συντελεστές.