Πανεπιστήμιο Κρήτης
Τμήμα Επιστήμης Υπολογιστών

ΗΥ-252 – Αντικειμενοστρεφής Προγραμματισμός
Βασίλης Χριστοφίδης

Τελική Εξέταση (3 ώρες)
Ημερομηνία: 1 Φεβρουαρίου 2009

Ονοματεπώνυμο:
Αριθμός Μητρώου:

Άσκηση 1(5 μονάδες) Αφαιρετικές Κλάσεις και κατασκευή αντικειμένων
Η παρακάτω κλάση `Person` έχει οριστεί σαν βασική κλάση (base class) για την σχεδίαση των υποκλάσεων `Student` και `Professor`. Η κλάση `Person` έχει δηλωθεί σαν αφαιρετική γιατί δεν θέλουμε να επιτρέπουμε την δημιουργία στιγμιοτύπων τύπου `Person`: το πρόγραμμα μπορεί να δημιουργεί μόνο στιγμιότυπα των κλάσεων `Student` και `Professor`.

```
public abstract class Person {
     private String name;
     private String ssn;
     public Person(String name, String ssn) {
          this.setName(name);
          this.setSsn(ssn);
     }
     // ... Rest of code omitted
}
```

Ωστόσο ο βασικός λόγος για την δήλωση μιας κλάσης σαν αφαιρετική είναι όταν η κλάση χρειάζεται να δηλώσει μεθόδους οι οποίες πρέπει/μπορούν να υλοποιηθούν μόνο από τις υποκλάσεις της. Αυτό δεν ισχύει για την κλάση `Person`. Πως θα μπορούσατε να διορθώσετε την παραπάνω δήλωση της `Person` ώστε:

1. Να μην είναι αφαιρετική κλάση.
2. Να μην μπορεί ένας κώδικας πελάτης να δημιουργήσει στιγμιότυπα της `Person`.
3. Να μπορεί ένας κώδικας πελάτης να δημιουργήσει στιγμιότυπα των υποκλάσεων `Student` και `Professor`.

Λύση:
The class Person must not be **abstract** and
the constructor should be made protected:

```
     protected Person(String name, String ssn) {
               this.setName(name);
               this.setSsn(ssn);
     }
```

Clients cannot instantiate a class with only protected constructors. Subclasses can still call super in their own constructors in order to invoke the protected constructors.

Άσκηση 2 (15 μονάδες) Βασική Λειτουργικότητα Αντικειμένων

Σας δίνεται η παρακάτω κλάση Java η οποία αναπαριστά ρητούς αριθμούς των οποίων ο αριθμητής και παρανομαστής είναι ακέραιοι αριθμοί.

```
public class Rational {
      protected int num, denom;
      public Rational(int num, int denom) {
            this.num = num;
            this.denom = denom;
            set(num, denom);
      }
      protected void set(int num, int denom) {
      if (denom == 0) throw new Error("Divide by zero");
            this.denom = denom;
            this.num = num;
      }
      public void reciprocate() {
            int oldDenom = this.denom;
            this.denom = num;
            this.num = oldDenom;
            set(denom, num);
      }
      public double toDouble() {
            return ((double) num / denom); }
      public String toString() { return num + "/" + denom; }
}
```

α) **(5 μονάδες)** Δεδομένης της default υλοποίησης των μεθόδων `hashCode()` και `equals(Object)`, τι πρόβλημα μπορεί να παρουσιαστεί στον παρακάτω κώδικα που δημιουργεί ένα σύνολο από ρητούς αριθμούς;

```
Set s = new HashSet();
s.add(new Rational(1,2));
s.add(new Rational(1,2));
```

Λύση:
The two Rational objects will each be viewed as distinct, because default hashCode is based on an object's address in memory. The Set which should have just once instance of the fraction would then have two instances.

β) **(10 μονάδες)** Υποσκελίστε την default υλοποίηση των μεθόδων `hashCode()` και `equals(Object)` ώστε να υποστηρίζουμε με συνεπή τρόπο συλλογές αντικειμένων τύπου `Rational` που βασίζονται σε συναρτήσεις κατακερματισμού.

Λύση:

```
public int hashCode() {
//Give credit for any reasonable implementation Best is:
  return num + 39*denom;
//alternatively return this.toString().hashCode();
}
public boolean equals(Object o) {
 if (o == null) return false;
 else if (!(o instance of Rational) return false;
 else {
  Rational r = (Rational)o;//you can assume this works
  return(this.num toDouble() == r.toDouble()num && this.denom == r.denom);} // or similar
}
```

Common Mistakes

a) Οι φοιτητές που απάντησαν λάθος στο συγκεκριμένο ερώτημα, θεωρούσαν ότι ουσιαστικά δεν υπήρχε λάθος καθώς η μέθοδος equals και hashCode της Object λειτουργούσαν κανονικά για τα αντικείμενα τύπου Rational χωρίς να χρειαστεί να τις υποσκελίσουν.

b) Η μέθοδος hashCode υλοποιήθηκε από το 5% περίπου των φοιτητών. Στη μέθοδο equals οι φοιτητές δεν την υποσκέλιζαν σωστά καθώς δεν τηρούσαν την υπογραφή της μεθόδου και δεν έκαναν του κατάλληλους ελέγχους για τις περιπτώσεις όπου το αντικείμενο ήταν null ή δεν ήταν στυγμιότυπο της κλασης Rational.

Άσκηση 3 (10 μονάδες) Βασική Λειτουργικότητα Αντικειμένων

Η Java ορίζει την διεπαφή `Comparable` ως εξής:

```
interface Comparable {
     int compareTo(Object o) ;
}
```

όπου `x.compareTo(y)` επιστρέφει έναν αρνητικό ακέραιο εάν x < y, 0 εάν είναι ίσα, και έναν θετικό ακέραιο εάν x > y. Ορίστε μία κλάση `MyArray` η οποία ενθυλακώνει πίνακες ακεραίων και υλοποιεί την διεπαφή `Comparable`. Στιγμιότυπα της `MyArray` πρέπει να συγκρίνονται χρησιμοποιώντας το άθροισμα (sum) των ακεραίων που περιέχουν οι ενθυλακωμένοι πίνακες. Για παράδειγμα, η εκτέλεση του παρακάτω τεστ

```
// create tow arrays and in initialize the elements
int [ ] a = new int [ ] { 1, 2, 3, 4 } ;
int [ ] b = new int [ ] {-1, 2, -3, 4, -5};
// create objects encapsulating these arrays
MyArray m1 = new MyArray(a) ;
MyArray m2 = new MyArray(b) ;
// print the result of the comparison
System.out.println (m1.compareTo(m2)) ;
```

θα τυπώσει έναν θετικό ακέραιο επειδή 1 + 2 + 3 + 4 > −1 + 2 − 3 + 4 − 5.

Υλοποιήστε πλήρως την κλάση MyArray ώστε ο παραπάνω κώδικας πελάτης να μπορεί να εκτελεστεί με επιτυχία.

Λύση:

```
public class MyArray implements Comparable {
 private int[ ] theArray ;
 private int theSum ;
 public MyArray (int[] intArr ) {
  theArray = intArr ;
  int sum = 0 ;
  for (int i=0; i < theArray.length ; i++) {
       sum = sum + theArray[i] ;
  }
  theSum = sum ;
 }
 public int getTheSum( ) {
  return theSum ;
 }
 public int compareTo(Object o) {
  int otherSum =((MyArray) o).getTheSum( ) ;
  return theSum - otherSum ;
 }
}
```

Common Mistakes

- Αρκετοί φοιτητές υλοποίησαν την μέθοδο compareTo παίρνοντας ως όρισμα ένα αντικείμενο τύπου MyArray,αντί για Object
- Ένα άλλο λάθος που έκαναν άλλοι είναι ότι δεν έκαναν casting το Object σε αντικείμενο MyArray, άρα στην συνέχεια δεν ήταν σωστό να προσπελάσουν τις μεθόδους της κλάσης MyArray, μέσω του ορίσματος (που είναι Object).
- Άλλο λάθος ήταν ότι κάποιοι φοιτητές δεν ενθυλάκωναν καθόλου τους πίνακες ακεραίων

Άσκηση 4 (28 μονάδες) Χειρισμός Πλαισίου Συλλογών Αντικειμένων

α) **(20 μονάδες)** Όταν χειριζόμαστε συλλογές τύπου HashMap, ορισμένες φορές χρειαζόμαστε να καθορίσουμε εάν δύο αντικείμενα HashMaps έχουν κοινά ζεύγη κλειδί/τιμή. Θεωρήστε για παράδειγμα τις παρακάτω συλλογές hashmap1 και hashmap2 που αντιστοιχούν συμβολοσειρές σε συμβολοσειρές (δηλ. ο τύπος τους είναι HashMap<String,String>) .

Key	Value
Αλίκη	Healthy
Μαρία	Ecstatic
Σπύρος	Happy
Βασίλης	Sick
Ειρήνη	Fine

hashmap1

Key	Value
Μαρία	Ecstatic
Βασίλης	Healthy
Θοδωρής	Superb
Μανώλης	Fine
Σπύρος	Happy

hashmap2

Τα κοινά ζεύγη στο παραπάνω παράδειγμα είναι: "Μαρία"/"Ecstatic" και "Σπύρος"/"Happy". Σημειώστε ότι αν και το κλειδί "Βασίλης" εμφανίζεται και στις δύο συλλογές, η συσχετισμένη τιμή με αυτό το κλειδί είναι διαφορετική στο hashmap1 από ότι στο hashmap2 (κατά συνέπεια δεν υπολογίζεται στα κοινά ζεύγη των δύο HashMaps). Παρόμοια, εάν μία τιμή εμφανίζεται και στις δύο συλλογές (π.χ. η τιμή "Fine") χωρίς όμως να συσχετίζεται με το ίδιο κλειδί δεν υπολογίζεται επίσης στα κοινά ζεύγη των δύο HashMaps. Σε αυτή την άσκηση θα πρέπει να υλοποιήσετε την μέθοδο :
public int commonKeyValuePairs(HashMap<String,String> map1, HashMap<String,String> map2)
η οποία δέχεται σαν ορίσματα δύο αντικείμενα τύπου HashMap<String,String> και επιστρέφει τον αριθμό των κοινών ζευγών κλειδί/τιμή των δύο HashMaps (για ευκολία υποθέστε ότι δεν έχουμε null τιμές).

Λύση:

```
/** Method: commonKeyValuePairs(map1, map2)
* Returns a count of the number of common key/value
* pairs in the two HashMaps that are passed in.
*/
public int commonKeyValuePairs(HashMap<String,String>
                    map1, HashMap<String,String> map2)
{
     int count = 0;
     // Get iterator over map1
     Iterator<String> it = map1.keySet().iterator();
     while (it.hasNext()) {
        // Get key from map1
        String key = it.next();
        // See if that keys exists in map2
        if (map2.containsKey(key)) {
   //alternatively
map1.get(key).equals(map2.get(key))
      //Lookup values associated with key in both
maps
           String map1Value = map1.get(key);
           String map2Value = map2.get(key);
           // See if values are equal
           if (map2Value.equals(map1Value)) {
               count++;
           }
        }
     }
     return count;
}
```

Εναλλακτικά:

```
public int commonKeyValuePairs(Map<String,String> map1,
Map<String, String> map2) {
```

```
        map1.entrySet().retainAll(map2.entrySet());
        return map1.size();
}
```

Common Mistakes: Αρκετοί φοιτητές– νόμιζαν ότι από HashMap μπορούν να πάρουν απευθείας έναν Iterator ο οπόιος είχε σαν πεδία το key και το value του κάθε στοιχείου του HashMap. Κάτι τέτοιο προφανώς δεν ισχύει. Επίσης το άλλο λάθος που έκαναν ήταν ότι προσπαθούσαν να διαχειριστούν το HashMap σαν πίνακα οπότε για κάθε i στοιχείο του προσπαθούσαν να πάρουν το key και το value του. Επίσης κάτι τέτοιο δεν ισχύει.

β) **(8 μονάδες)** Σε συλλογές τύπου `HashMap` or `HashSet` συνήθως εκτελούμε αποδοτικά πολλές πράξεις όπως εισαγωγές (add), διαγραφές (remove), αναζητήσεις (contains) ή άλλες. Οι επιδόσεις αυτών των πράξεων εξαρτώνται από τις ιδιότητες που έχει η συνάρτηση κατακερματισμού (hash function) καθώς και ο παράγοντας φόρτωσης (load factor).

(i) Ποια ιδιότητα πρέπει να έχει η συνάρτηση κατακερματισμού ώστε αυτές οι πράξεις να εκτελούνται πολύ αποδοτικά;

Λύση:
A hash function converts a key value into an integer that is used to select the bucket (or slot) where that key should be stored. For hashing to be efficient, the hash function must distribute the keys among the buckets uniformly so no one bucket winds up with too many key values.

(ii) Ποια ιδιότητα πρέπει να έχει ο παράγοντας φόρτωσης ώστε αυτές οι πράξεις να εκτελούνται πολύ αποδοτικά;

Λύση:
The load factor is the ratio n/b of the number of items in the table (n) to the total number of buckets (b). For hashing to be efficient, this number needs to be small.

Άσκηση 5 (28 μονάδες) Σαρωτές Συλλογών Αντικειμένων

Σε αυτή την άσκηση θα υλοποιήσετε ορισμένους σαρωτές (Iterators) χρησιμοποιώντας άλλους που προσφέρονται από το πλαίσιο συλλογών αντικειμένων της Java. Στις υλοποιήσεις σας θα πρέπει να λάβετε υπόψη τις εξαιρέσεις που ενδεχομένως να δημιουργηθούν.

α) (**10 μονάδες**) Υλοποιήστε έναν σαρωτή `ReverseIterator` ο οποίος επιστρέφει τα στοιχεία (τύπου `Object`) μιας λίστας (τύπου `List`) σε ανεστραμμένη σειρά. Η λίστα δίνεται σαν όρισμα στην μέθοδο κατασκευής (constructor) του σαρωτή. Ο σαρωτής θα πρέπει να υλοποιεί όλη την λειτουργικότητα της διεπαφής `Iterator`, δηλ τις μεθόδους `hasNext()`,

next() και remove(), χρησιμοποιώντας κατάλληλες μεθόδους της διεπαφής List όπως size(), get(int) και remove(int).

Λύση:

```
import java.util.*;
public class ReverseIterator implements Iterator {
     private ListIterator listit; // 2 points

     public ReverseIterator(List list) {
        this.listit =
                        listit.listIterator(list.size())
;
     }// 2 points
     public boolean hasNext() {
        return listit.hasPrevious();
     }// 2 points
     public Object next() {
        return listit.previous();
     }// 2 points
     public void remove() {
        listit.remove();
     }// 2 points
}
```

Common Mistakes: Αρκετοί ήταν οι φοιτητές που αντί για μια κλάση με τα παραπάνω χαρακτηριστικά έφτιαξαν μια μέθοδο που επέστρεφε μια λίστα με ανεστραμμένα τα στοιχεία.
Επίσης κοινό λάθος ήταν η δημιουργία στον κατασκευαστή μιας λίστας με ανεστραμμένα τα στοιχεία.

β) (**18 μονάδες**) Υλοποιήστε έναν σαρωτή ZipperIterator πού συναρμόζει (“zips") τα στοιχεία (τύπου Object) που επιστρέφουν δύο άλλοι σαρωτές που δίνονται σαν ορίσματα στην μέθοδο κατασκευής (constructor) του ZipperIterator. Πιο συγκεκριμένα, εάν ο πρώτος σαρωτής επιστρέφει τα στοιχεία με την σειρά x0; x1; … ; xn; Και ο δεύτερος σαρωτής επιστρέφει τα στοιχεία με την σειρά y0; y1; …; yn;, ο σαρωτής ZipperIterator θα επιστρέφει τα στοιχεία με την σειρά x0; y0; x1; y1;…; xn; yn. Με άλλα λόγια, ο ZipperIterator επιστρέφει εναλλακτικά τα στοιχεία από τους σαρωτές που του δίνονται σαν ορίσματα. Ο ZipperIterator τερματίζει την σάρωση (δηλ. η hasNext γίνεται ψευδής) όταν ένας από τους δύο σαρωτές εισόδου εξαντλεί τα στοιχεία της συλλογής που σαρώνει. Η μέθοδος remove() του ZipperIterator διαγράφει το πιο πρόσφατα σαρωμένο στοιχείο από τον σαρωτή που το έχει επιστρέψει. Σας δίνεται στην συνέχεια ένας κώδικας πελάτης που επιδεικνύει την λειτουργία του ZipperIterator:

```
import java.util.*;
List l = new ArrayList();
List r = new ArrayList();
l.add("a");
l.add("b");
l.add("c");
```

```
r.add("A");
Iterator z = new ZipperIterator(l.iterator(),r.iterator());
while (z.hasNext()) {
     System.out.println(z.next());
     z.remove();
     System.out.println(l);
     System.out.println(r);
     }

Output
a
[b, c]
[A]
A
[b, c]
[]
b
[c]
[]
```

Βοήθεια: Στην υλοποίηση του ZipperIterator θα χρειαστείτε μια μεταβλητή στιγμιοτύπων (πχ. nextIsLeft) η οποία υποδεικνύει ποιος από τους δύο σαρωτές που δίνονται σαν ορίσματα θα προσφέρει το επόμενο στοιχείο που θα επιστραφεί.

Λύση:

```
import java.util.*;
public class ZipperIterator implements Iterator {
     private Iterator left, right;
     private boolean nextIsLeft; // 3 points
     public ZipperIterator(Iterator left,
                           Iterator right) {
      this.left = left;
      this.right = right;
      nextIsLeft = true;
     } // 4 points
     public boolean hasNext() {
     return nextIsLeft?
left.hasNext():right.hasNext();
     } // 2 points
     public Object next() {
      E next;
      next = nextIsLeft? left.next():right.next();
      nextIsLeft = !nextIsLeft;
      return next;
     } // 5 points
     public void remove() {
      if (nextIsLeft)
          right.remove();
      else
          left.remove();
     }// 4 points
}
```

Common Mistakes: Αρκετοί ήταν οι φοιτητές που αντί για μια κλάση με τα παραπάνω χαρακτηριστικά έφτιαξαν μια μέθοδο που επέστρεφε μια λίστα.

Κοινό λάθος ήταν επίσης η δημιουργία μιας λίστας στον κατασκευαστή με τα στοιχεία των Iterators εναλλακτικά.

Τέλος πολλοί ήταν αυτοί που θεώρησαν ότι τερματίζει ο zipperIterator μόλις ένας από τους δυο Iterators τερματίζει:

return it1.hasNext() && it2.hasNext() .

Άσκηση 6 (15 μονάδες) Γενικοί Τύποι στην Java

α) (**5 μονάδες**) Η Java προσφέρει περισσότερους από έναν τρόπους για να σαρώσουμε τα στοιχεία μιας λίστας. Για παράδειγμα σε μία λίστα από συμβολοσειρές:

```
List<String> list = new LinkedList<String>();
list.add("I");
list.add("love");
list.add("everybody");
```

μπορούμε να σαρώσουμε τα στοιχεία της λίστας `list` με τον κώδικα:

```
for (String s: list) // Technique 1
  System.out.println(s);
```

ή εναλλακτικά με τον κώδικα:

```
Iterator<String> i = list.iterator(); // Technique 2
while (i.hasNext())
  System.out.println(i.next());
```

Υπάρχει κάποια διαφορά στα στοιχεία της λίστας που επιστρέφουν οι παραπάνω κώδικες? Δώστε μια σύντομη και περιεκτική εξήγηση.

Λύση:
There is no difference. The first technique is easier to write. The second technique, using the `Iterator` class, is usually used only when you need to do something that is possible only with an `Iterator`, such as selectively delete list members during iteration.

β) (**5 μονάδες**) Δεδομένης της παρακάτω δήλωσης της κλάσης `Derived`

```
public class Derived extends Base { ...
```

είναι ο γενικός τύπος `LinkedList<Derived>` ένας υποτύπος του `LinkedList<Base>`? Δώστε μια σύντομη και περιεκτική εξήγηση.

Λύση:
`LinkedList<Derived>` is not a subclass of `LinkedList<Base>`. If B is a subclass of A, then instances of B should be substitutable for instances of A. However, if a is a `LinkedList<Base>`, one can perform operations such as `a.add(new Base());`

This operation would not be permitted for a `LinkedList<Derived>`. Consequently, a `LinkedList<Derived>` is not substitutable for a `LinkedList<Base>`. In other words, you cannot use a `LinkedList<Derived>` where a `LinkedList<Base>` is expected.

γ) (**5 μονάδες**) Σε ένα πρόγραμμα Java θα πρέπει να μπορεί να εκλεχθεί στατικά (δηλ. κατά την μετάφραση) η ορθότητα των τύπων όλων των εκφράσεων που χρησιμοποιεί. Π.χ αντί για την δήλωση της παραμέτρου <T>, θα πρέπει να γράψουμε την πιο περιορισμένη: `<T extends ...>`.
Πως θα μπορούσατε να συμπληρώσετε τον παρακάτω κώδικα (στην υπογραμμισμένη περιοχή) έτσι ώστε να είναι ένα σωστό ως προς τους τύπους πρόγραμμα Java; Δώστε μια σύντομη και περιεκτική εξήγηση για την συμπλήρωση πού προτείνετε περιγράφοντας όλες τις επιπλέον κλάσεις ή διεπαφές που σας είναι απαραίτητες. Προσπαθήστε να κάνετε την υπογραφή της μεθόδου όσο περισσότερο γενική μπορείτε στα πλαίσια όμως της ορθότητας.

```
class C {
     ...
     <T extends _____> int f (T x, T y){
          ... x.compareTo(y) ...
     }
     ...
}
```

Λύση:
We call method `compareTo()` on T, which is defined in `Comparable`, so T must be a subtype of `Comparable`, therefore its an upper bound: `<T extends Comparable>`. `Comparable` has also a generic version (say with E) and it define its method as "`compareTo(E e)`". We only pass arguments of type T to this method, so E must assignable from T, i.e. a supertype. Thus, T is a lower bound, and we need `<? super T>`. All together: `<T extends Comparable<? super T>>`
`<T extends Comparable>` is also correct, since using the raw interface `Comparable` makes the argument of "`compareTo(E e)`" an `Object`, thus assignable from any possible reference T. Generally speaking, we would avoid raw types, because the compiler only issues a warning and then drops all generic type-checking on all usages of the raw type.

Common Mistakes: 6α) Πολλοί φοιτητές είπαν ότι στην τεχνική 1 το s που εκτυπώνεται είναι ένα String ενώ στην τεχνική 2 το i.next() είναι Object.
Κάποιοι άλλοι είπαν ότι στην τεχνική 1 δημιουργείται ένα String με όλα τα περιεχόμενα της λίστας.
Τέλος κάποιοι είπαν ότι στην τεχνική 1 το τελευταίο στοιχείο που θα εκτυπωθεί θα είναι το null και ότι στη τεχνική 2 η εκτύπωση θα αρχίσει από το δεύτερο στοιχείο.

_

6β) Το πιο κοινό λάθος ήταν ότι θεωρούσαν ότι ο τύπος LinkedList<Derived> είναι υπότυπος του LinkedList<Base>, επειδή είναι μια λίστα από αντικείμενα τύπου Derived και ο τύπος Derived είναι subclass of Base.

Άσκηση 7 (8 μονάδες) Θεωρία Αντικειμενοστρεφούς Κώδικα

α) **(4 μονάδες)** Για ποιους λόγους θα χρησιμοποιούσατε μια διεπαφή Java στην σχεδίαση μιας ιεραρχίας κλάσεων;

Λύση:
In general, the base class in a class hierarchy has two main purposes. First, it defines the common interface for using the set of classes. Second, because subclasses inherit methods, the base class is the proper place to implement code that will be common to all subclasses. Although the base class does specify the interface, it also includes implementation information that is of no interest to clients. The usual practice is to use a Java Interface to specify the interface, and then have the base class implement that interface. Client code is typically written to the Java Interface. This not only simplifies the task of writing client code, but the resulting code is more flexible because it will also work with different implementations of the Interface.

β) **(4 μονάδες)** Τι είναι ένας τύπος Java; Τι είναι οι πρωτογενείς (primitive) τύποι; Τι είναι οι τύποι που ορίζονται από τούς χρήστες (user-defined); Πώς η Java χρησιμοποιεί τους τύπους ώστε να διευκολύνει τον προγραμματισμό και να καθιστά τα προγράμματα πιο αξιόπιστα;

Λύση:
A type is a set of values and operations that may be performed on them. For instance int is a 32-bit integer type, with values ranging from -2^{31} to $2^{31}-1$. Valid operations on int are addition, subtraction, multiplication, division, the bitwise operations, etc. Java's primitive types are: int, byte, short, long, float, double, char and boolean. A user-defined type is a composition of built-in types with special operations defined on it. In Java, a user-defined type is called a class or an interface. Java checks to make sure every operation is valid with respect to its operands' types at compile time. This prevents careless errors, like treating an integer value as a floating-point, or vice versa. Type checking ensures that every operation in the program is conceptually valid before the program has a chance to run and crash.