Data Structures and Algorithms in Java™

Sixth Edition

Michael T. Goodrich
Department of Computer Science
University of California, Irvine

Roberto Tamassia
Department of Computer Science
Brown University

Michael H. Goldwasser
Department of Mathematics and Computer Science
Saint Louis University

Instructor's Solutions Manual

WILEY

Chapter 1 Java Primer

Hints and Solutions

Reinforcement

R-1.1) Hint Use the code templates provided in the Simple Input and Output section.

R-1.2) Hint You may read about cloning in Section 3.6.

R-1.2) Solution Since, after the clone, *A*[4] and *B*[4] are both pointing to the same GameEntry object, *B*[4].score is now 550.

R-1.3) Hint The modulus operator could be useful here.

R-1.3) Solution

```
public boolean isMultiple(long n, long m) {
  return (n%m == 0);
}
```

R-1.4) Hint Use bit operations.

R-1.4) Solution

```
public boolean isEven(int i) {
  return (i & 1 == 0);
}
```

R-1.5) Hint The easy solution uses a loop, but there is also a formula for this, which is discussed in Chapter 4.

R-1.5) Solution

```
public int sumToN(int n) {
  int total = 0;
  for (int j=1; j <= n; j++)
    total += j;
  return total;
}
```

R-1.6) Hint The easy thing to do is to write a loop.

R-1.6) Solution

```
public int sumOdd(int n) {
  int total = 0;
  for (int j=1; j <= n; j += 2)
    total += j;
  return total;
}
```

R-1.7) Hint The easy thing to do is to write a loop.

R-1.7) Solution

```
public int sumSquares(int n) {
  int total = 0;
  for (int j=1; j <= n; j++)
    total += j*j;
  return total;
}
```

R-1.8) Hint You might use a switch statement.

R-1.8) Solution

```
public int numVowels(String text) {
  int total = 0;
  for (int j=0; j < text.length( ); j++) {
    switch (text.charAt(j)) {
    case 'a':
    case 'A':
    case 'e':
    case 'E':
    case 'i':
    case 'I':
    case 'o':
    case 'O':
    case 'u':
    case 'U':
      total += 1;
    }
  }
  return total;
}
```

R-1.9) Hint Consider each character one at a time.

R-1.10) Hint Consider using get and set methods for accessing and modifying the values.

R-1.11) Hint The traditional way to do this is to use setFoo methods, where Foo is the value to be modified.

R-1.11) Solution

```
  public void setLimit(int lim) {
    limit = lim;
  }
```

R-1.12) Hint Use a conditional statement.

R-1.12) Solution

```
  public void makePayment(double amount) {
    if (amount > 0)
      balance -= amount;
  }
```

R-1.13) Hint Try to make wallet[1] go over its limit.

R-1.13) Solution

```
for (int val=1; val <= 58; val++) {
  wallet[0].charge(3*val);
  wallet[1].charge(2*val);
  wallet[2].charge(val);
}
```

This change will cause wallet[0] to attempt to go over its limit.

Creativity

C-1.14) Hint The Java method does not need to be passed the value of n as an argument.

C-1.15) Hint Note that the Java program has a lot more syntax requirements.

C-1.16) Hint Create an enum type of all operators, including $=$, and use an array of these types in a switch statement nested inside for-loops to try all possibilities.

C-1.17) Hint Note that at least one of the numbers in the pair must be even.

C-1.17) Solution

```
public boolean hasEvenPair(int[ ] data) {
  if (data.length > 1) {
    for (int j=0; j < data.length; j++)
      if (data[j] % 2 == 0)
        return true;
  }
  return false;
}
```

C-1.18) Hint Use the Math.pow function for calculations. Use your solution for norm(v,p) to implement norm(v).

C-1.18) Solution

```
public double norm(double[ ] v, int p) {
  int total = 0;
  for (double k : v)
    total += Math.pow(k,p);
  double exp = 1.0/p;
  return Math.pow(total, exp);
}

public double norm(double[ ] v) {
  return norm(v,2);
}
```

C-1.19) Hint This is the same as the logarithm, but you can use recursion here rather than calling the log function.

C-1.20) Hint The simple solution just checks each number against every other one, but we will discuss better solutions later in the book. Make sure you don't compare a number to itself.

C-1.20) Solution

```
public boolean distinct(float[ ] data) {
  for (int j=0; j < data.length − 1; j++)
    for (int k=j+1; k < data.length; k++)
      if (data[j] == data[k])
        return false;
  return true;
}
```

C-1.21) Hint Consider using swaps to reshuffle the array one entry at a time, starting from the beginning and moving to the end.

C-1.22) Hint There are many solutions. If you know about recursion, the easiest solution uses this technique. Otherwise, consider using an array to hold solutions. If this still seems to hard, then consider using six nested loops (but avoid repeating characters and make sure you allow all string lengths).

C-1.22) Solution Here is a possible solution:

```
public static void permute(ArrayList bag, ArrayList permutation) {
  // When the bag is empty, a full permutation exists
  if (bag.isEmpty( ) ) {
    System.out.println(permutation);
  } else {
    // For each element left in the bag
    for(int i = 0; i < bag.size( ); i++) {
      // Take the element out of the bag
      // and put it at the end of the permutation
      Object obj = bag.get(i);
      bag.remove(i);
      permutation.add(obj);

      // Permute the rest of the bag (recursively)
      permute(bag, permutation);

      // Take the element off the permutation
      // and put it back in the bag
      permutation.remove(permutation.size( ) − 1);
      bag.add(i, obj);
    }
  }
}

public static void main(String[ ] args) {
  ArrayList<Character> orig = new ArrayList<>( );
  char[ ] word = {'c', 'a', 't', 'd', 'o', 'g'};
  for (char c : word)
    orig.add(c);
  permute(orig, new ArrayList( ));
}
```

C-1.23) Hint Go back to the definition of dot product and write a for loop that matches it.

C-1.23) Solution

```
public int[ ] compute(int[ ] a, int[ ] b) {
  if (a.length != b.length)
    throw new IllegalArgumentException("arrays must have same length")
  int[ ] c = new int[a.length];
  for (int j=0; j < a.length; j++)
    c[j] = a[j] * b[j];
  return c;
}
```

C-1.24) Hint The card is no longer needed as an explicit parameter.

C-1.24) Solution

```
public void printSummary( ) {
  System.out.println("Customer = " + customer);
  System.out.println("Bank = " + bank);
  System.out.println("Account = " + account);
  System.out.println("Balance = " + balance);
  System.out.println("Limit = " + limit);
}
```

C-1.25) Hint You might use a StringBuilder to compose the pieces of the string into one large string (including newlines).

C-1.25) Solution

```
public String toString( ) {
  StringBuilder sb = new StringBuilder( );
  sb.append("Customer = " + customer + System.lineSeparator( ));
  sb.append("Bank = " + bank + System.lineSeparator( ));
  sb.append("Account = " + account + System.lineSeparator( ));
  sb.append("Balance = " + balance + System.lineSeparator( ));
  sb.append("Limit = " + limit + System.lineSeparator( ));
  return sb.toString( );
}
```

Projects

P-1.26) Hint Use an array to buffer all the original lines.

P-1.27) Hint You do not need to use a graphical user interface, but you may want to use the System.console() method.

P-1.28) Hint Define a way of indexing all the sentences and the location in each one and then work out a way of picking eight of these locations for a typo.

P-1.29) Hint Use a two-dimensional array to keep track of the statistics and a one-dimensional array for each experiment.

P-1.30) Hint We recommend using the Java Swing package.