Practice Exam #1

SECTION I

Time — 1 hour and 30 minutes
Number of questions — 40
Percent of total grade — 50

1. Consider the following code segment.

```
int a = 1;
int b = 2;
double c = a/b;
double d = (double)(a/b);
System.out.println(c + d);
```

What is printed when the code segment is executed?

(A) 0
(B) 0.0
(C) 0.5
(D) 1.0
(E) 1.5

2. Consider the following method and the code segment in the same class.

```
public static void process(String word)
{
  word = word.substring(2) + word;
}

  String word = "test";
  process(word);
  System.out.println(word.indexOf("t"));
```

What is printed when the code segment is executed?

(A) -1
(B) 0
(C) 1
(D) 2
(E) 3

3. Consider the following code segment.

```
int sum = 0;
for (int k = 1; k <= 10; k += k)
{
  sum += k;
}
System.out.println(sum);
```

What is printed when the code segment is executed?

(A) 10
(B) 14
(C) 15
(D) 25
(E) 55

4. Consider the following statement.

```
System.out.println("\"No\nSmoking\"".indexOf("S"));
```

What is printed when the statement is compiled and executed?

(A) Nothing, due to a syntax error
(B) 4
(C) 5
(D) 6
(E) 7

5. Consider the following method.

```
public int mystery(int n)
{
  if (n == 1)
    return 3;
  else
    return 3*mystery(n-1) + 1;
}
```

What does `mystery(4)` return?

(A) 10
(B) 31
(C) 94
(D) 281
(E) 283

6. Consider the following method with a mistake.

```
public static void respondTo(String str)
{
  int f = str.indexOf("FAVORITE");
  int b = str.indexOf("BOOK");
  int m = str.indexOf("MOVIE");

  if (f >= 0)
    if (b > f)
      System.out.println("IT IS MY FAVORITE BOOK, TOO");
    else if (m > f)
      System.out.println("IT IS MY FAVORITE MOVIE, TOO");
  else if (str.indexOf("NAME") > 0)
    System.out.println("MY NAME IS CHITCHAT");
}
```

It doesn't work as intended, because a pair of braces are missing.

If

```
str = "THE JUNGLE BOOK MOVIE " +
                "IS BASED ON THE BOOK BY THE SAME NAME.";
```

what is printed when `respondTo(str)` is called?

(A) `Nothing`

(B) `MY NAME IS CHITCHAT`

(C) `IT IS MY FAVORITE BOOK, TOO`

(D)
```
IT IS MY FAVORITE BOOK, TOO
MY NAME IS CHITCHAT
```

(E)
```
IT IS MY FAVORITE BOOK, TOO
IT IS MY FAVORITE MOVIE, TOO
MY NAME IS CHITCHAT
```

7. Which of the following statements DOES NOT print `1976`?

(A) `System.out.println("1976");`
(B) `System.out.println(19 + 76 + "");`
(C) `System.out.println("" + 19 + 76);`
(D) `System.out.println("19" + 76);`
(E) `System.out.println("19" + "76");`

8. What is the result when the following code segment is compiled and executed?

```
Double x = 2.0;                    // Line 1
double y = Math.pow(x, 3);         // Line 2
Integer p = (int)(y + 0.5);        // Line 3
System.out.println(p);
```

(A) Incompatible types error on Line 1
(B) Incompatible types error on Line 2
(C) Incompatible types error on Line 3
(D) `8` is displayed
(E) `9` is displayed

9. Consider the following code segment.

```
int a = 3;
int b = a % 3;
a += b;
if (a > 3 && a/b < 2)
  System.out.println(a/b);
```

What is printed when the code segment is executed?

(A) The value of `Integer.MAX_VALUE`
(B) `Infinity`
(C) `java.lang.ArithmeticException: / by zero`
(D) `3`
(E) Nothing

10. Which of the following expressions evaluate to `true`?

I. `"Tick-Tock".compareTo("Tick" + "-" + "Tock") == 0;`
II. `"Tick-Tock".compareTo("Tock-Tick") < 0;`
III. `"Tick-Tock".indexOf("T") > "Tick-Tock".indexOf("t");`

(A) None of the three
(B) I only
(C) I and II only
(D) II and III only
(E) I, II, and III

11. Assume that `x` and `y` are `boolean` variables that have been properly declared and initialized. When do the logical expressions `!x && (x || y)` and `x || !y` have the same value?

(A) Never
(B) Only when `x` is `true`
(C) Only when `x` is `false`
(D) Only when both `x` and `y` are `true`
(E) Only when both `x` and `y` are `false`

12. Recall that the area of a triangle with base b and height h is $\frac{b \cdot h}{2}$. Suppose the `double` variables `b` and `h` are declared and initialized to positive values. Which of the following statements evaluates to the area of a triangle with base `b` and height `h`, rounded to the nearest integer?

(A) `(int)((0.5*b*h);`
(B) `(int)((b*h + 1)/2));`
(C) `(int)((b*h + 0.5)/2);`
(D) `(int)((b*h - 0.5)/2);`
(E) `(int)(0.5*b) * (int)(0.5*h) / 2;`

13. Given the 2D array

```
int[][] magic = {{4, 9, 2},
                 {3, 5, 7},
                 {8, 1, 6}};
```

what is the value of `sum` after the following code segment is executed?

```
int sum = 0;

for (int r = 0; r < magic.length; r++)
{
  for (int c = r; c < magic[r].length; c++)
  {
    sum += magic[r][c];
  }
}
```

(A) 30
(B) 31
(C) 33
(D) 35
(E) 45

14. What is the most likely value of the expression

```
(int)(Math.random() + Math.random() + 0.5)
```

(A) 0
(B) 1
(C) 2
(D) 1 and 2 are equally likely; 0 is less likely
(E) 0, 1, and 2 are equally likely

15. Consider the following method.

```
public static String fiboDigits(int n)
{
  int[] fibs = {1, 2, 3, 5, 8, 13, 21, 34};
  String s = "";

  for(int i = fibs.length - 1; i >= 0; i--)
  {
    if (n >= fibs[i])
    {
      s += "1";
      n -= fibs[i];
    }
    else
    {
      s += "0";
    }
  }

  return s;
}
```

Which value does `fiboDigits(30)` return?

(A) `"1010001"`
(B) `"1100001"`
(C) `"01001001"`
(D) `"01010001"`
(E) `"01100001"`

16. What is the value of `count` after the following code segment is executed?

```
int count = 0;
for (int j = 1; j <= 2; j++)
{
  count++;
  for (int k = 1; k <= 3; k++)
    count *= 2;
}
```

(A) 3
(B) 8
(C) 17
(D) 72
(E) 272

17. Consider the following code segment.

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(0);
numbers.add(-1);
numbers.add(1);
numbers.add(-2);
numbers.add(-3);
int i = 0;
for (Integer x : numbers)
{
  if (x < 0)
    numbers.remove(i);
  i++;
}
System.out.println(numbers);
```

What is the outcome when this code is compiled and executed?

(A) `ConcurrentModificationException`
(B) `IndexOutOfBoundsException`
(C) The code runs with no errors, `[0, 1]` is printed
(D) The code runs with no errors, `[0, 1, -3]` is printed
(E) The code runs with no errors, `[0, -1, 1, -2, -3]` is printed

18. What is printed when the following code segment is executed?

```
ArrayList<Integer> list1 = new ArrayList<Integer>();
ArrayList<Integer> list2 = list1;
list1.add(1);
list2.add(0, 2);
System.out.println(list1 + " " + list2);
```

(A) `[1] [0]`
(B) `[1] [2]`
(C) `[1, 2] [2]`
(D) `[1] [0, 2]`
(E) `[2, 1] [2, 1]`

19. Consider the following class.

```
public class Test
{
  private int t = 0;

  public Test(int x)
  {
    int t = x;
  }

  public int get()
  {
    return t;
  }

  public int increment()
  {
    t++;
    return t;
  }
}
```

What is displayed when the following code segment in a client class of `Test` is executed?

```
Test test = new Test(3);
int x = test.increment();
System.out.println(x + test.get());
```

(A) 0
(B) 1
(C) 2
(D) 3
(E) 5

20. Consider the following method.

```
public String fun(String s)
{
  if (s.length() <= 2)
    return s;
  else
    return fun(s.substring(2)) + s.substring(0, 2);
}
```

What is displayed by `System.out.println(fun("ABCDE"))`?

(A) ABCDE
(B) CDEAB
(C) CDABE
(D) ECDAB
(E) DECAB

21. Consider the following method.

```
/** Returns the index of target in arr or -1 if not found.
 *  Precondition: The elements of arr are sorted
 *                in ascending order; arr contains
 *                no duplicates.
 */
public static int find(String[] arr, String target)
{
  < code not shown >
}
```

Suppose an optimal search algorithm is used in the above `find` method. Suppose also that `names.length` is 5 and `names` satisfies `find`'s precondition. What is the worst-case number of calls to `String`'s `compareTo` method when `find(names, "Desiree")` is called?

(A) 1
(B) 2
(C) 3
(D) 4
(E) 5

22. Consider the following class `Person` and its subclass `Student` with missing code.

```
public class Person
{
  private String name;

  public Person(String nm)
  {
    name = nm;
  }

  public String toString()
  {
    return name;
  }
}

public class Student extends Person
{
  private int grade;

  public Student(String nm, int gr)
  {
    < missing statement >
    grade = gr;
  }

  public String toString()
  {
    < code not shown >
  }
}
```

The statements

```
Person esti = new Student("Esther", 11);
System.out.println(esti);
```

in a client class compile with no errors and display

```
Esther, grade 11
```

Which of the following could replace < *missing statement* > to make this happen?

(A) `super(nm);`
(B) `name = nm;`
(C) `super.name = nm;`
(D) `this = Person(name);`
(E) No replacement will make this work, regardless of the code that is not shown.

23. The method

```
public static boolean testSumOdds(int n)
{
  < missing code >
}
```

tests the hypothesis that $1+3+5+...+(2k-1)=k^2$ for all positive integers k. This method returns `true` if the formula holds for k from 1 to `n` and returns `false` if the formula fails for some k in that range. Which of the following could replace *< missing code >*?

I.
```
for (int k = 1; k <= n; k++)
{
  int sum = 0;
  for (int j = 1; j < 2*k; j += 2)
    sum += j;

  if (sum != k*k)
    return false;
}
return true;
```

II.
```
int sum = 0;

for (int k = 1; k <= n; k++)
{
  sum += 2*k-1;

  if (sum != k*k)
    return false;
}
return true;
```

III.
```
int sum = 0;

for (int k = 1; k <= n; k += 2)
{
  sum += k;

  if (sum != k*k)
    return false;
}
return true;
```

(A) I only
(B) II only
(C) I and II only
(D) II and III only
(E) I, II and III

24. Consider the following class `Text` and its subclass `SubText`.

```
public class Text
{
  private String word = "";

  public Text() { word = "A"; }

  public void append(String s) { word += s; }
  public String toString() { return word; }
}

public class SubText extends Text
{
  public SubText() { append("B"); }

  public void append(String s) { super.append("-" + s); }
}
```

What is printed when the following code segment in some other class is executed?

```
Text t = new SubText();
t.append("C");
System.out.println(t);
```

(A) `AC`
(B) `A-C`
(C) `ABC`
(D) `-B-C`
(E) `A-B-C`

25. For which values of the `boolean` variables `x` and `y` does the expression

```
(!(x && y) == (!x || !y)) && (!(x || y) == (!x && !y))
```

evaluate to `true`?

(A) For any values
(B) Only when both `x` and `y` are `true`
(C) Only when both `x` and `y` are `false`
(D) Only when `x` and `y` have different values
(E) Never

26. Consider the following method.

```
public static boolean isGood(int[][] m)
{
  for (int r = 0; r < m.length; r++)
  {
    if (r > 0 && m[r][0] < m[r-1][0])
    {
      return false;
    }

    for (int c = 1; c < m[0].length; c++)
    {
      if (m[r][c] < m[r][c-1])
      {
        return false;
      }
    }
  }
  return true;
}
```

For which of the following 2D arrays `m` `isGood(m)` returns `true`?

I.
```
int[][] m = {{1, 3, 5, 7},
             {2, 3, 4, 8},
             {3, 4, 5, 6}};
```

II.
```
int[][] m = {{5, 10, 15, 20},
             {10, 15, 20, 25},
             {11, 12, 13, 14}};
```

III.
```
int[][] m = {{0, 1, 5, 6},
             {1, 2, 3, 4},
             {3, 2, 1, 0}};
```

(A) None of the three
(B) I only
(C) I and II only
(D) II and III only
(E) I, II, and III

27. Consider the following class `Location` and a code segment in another class.

```
public class Location
{
  private int row, col;

  public Location(int r, int c) { row = r; col = c; }

  public Location(Location other)
  {
    row = other.row;    // Line *
    col = other.col;
  }

  public void move(int dr, int dc) { row += dr; col += dc; }

  public String toString()
  {
    return "(" + row + ", " + col + ")";
  }
}
```

```
Location loc = new Location(0, 1);
Location[] list = {loc, loc, new Location(loc)};

loc.move(10, 10);    // Line **

for (Location xy : list)
  System.out.print(xy + " ");
```

What is the outcome when this code is compiled and executed?

(A) Syntax error in `Location`'s constructor on Line *
(B) `ConcurrentModificationException` thrown at run time on Line **
(C) `(10, 11) (0, 1) (0, 1)` is displayed
(D) `(10, 11) (10, 11) (0, 1)` is displayed
(E) `(10, 11) (10, 11) (10, 11)` is displayed

28. The statement

```
double x = samples[ < missing index > ];
```

assigns to `x` an element of the array `samples`, randomly chosen with equal probabilities. Which of the following can replace < *missing index* >?

(A) `(int)(samples.length * Math.random())`
(B) `(int)(samples.length * Math.random()) - 1`
(C) `(int)(samples.length * Math.random()) + 1`
(D) `(int)(samples.length * Math.random() + 0.5)`
(E) `samples.length * (int)(Math.random() + 0.5)`

29. Suppose the method `swap(int[] arr, int i, int j)` swaps `arr[i]` and `arr[j]`. Which of the following code segments (in the same class) will always arrange the first three elements in `arr` in ascending order, assuming these values are all different?

(A)
```
if (arr[0] > arr[1])
  swap(arr, 0, 1);
if (arr[2] < arr[1])
  swap(arr, 1, 2);
```

(B)
```
if (arr[0] > arr[1])
  swap(arr, 0, 1);
if (arr[2] < arr[0])
  swap(arr, 0, 2);
```

(C)
```
if (arr[0] > arr[1])
{
  swap(arr, 0, 1);
}
else
{
  if (arr[2] < arr[1])
    swap(arr, 0, 2);
  else
    swap(arr, 0, 1);
}
```

(D)
```
if (arr[0] > arr[1])
  swap(arr, 0, 1);
if (arr[2] < arr[0])
  swap(arr, 0, 2);
if (arr[2] > arr[1])
  swap(arr, 1, 2);
```

(E) None of the above

Questions 30 and 31 refer to the following method.

```
public static int getBellNumber(int n)
{
  int[] row = {1};

  for (int k = 1; k < n; k++)
  {
    int[] nextRow = new int[row.length + 1];
    nextRow[0] = row[row.length - 1];

    for (int i = 1; i <= k; i++)
    {
      nextRow[i] = nextRow[i-1] + row[i-1];  // Line X
    }

    row = nextRow;
  }
  return row[row.length - 1];
}
```

30. What does `getBellNumber(5)` return?

(A) 15
(B) 37
(C) 52
(D) 151
(E) 203

31. How many times is the statement on Line X executed when `getBellNumber(5)` is called?

(A) 5
(B) 10
(C) 15
(D) 16
(E) 20

32. Consider the following class `Good` and its subclass `VeryGood`.

```
public class Good
{
  public Good(int t)
  {
    while (t > 0)
    {
      doSomething();
      t--;
    }
  }

  public void doSomething()
  {
    System.out.println("This is good");
  }
}

public class VeryGood extends Good
{
  public void doSomething()
  {
    System.out.println("This is very good");
  }

  public void doSomething(int t)
  {
    while (t > 0)
    {
      doSomething();
      t--;
    }
  }
}
```

The statements

```
Good x = new VeryGood();
x.doSomething();
```

in another class, when compiled, report a syntax error. Which of the following might cause that error?

(A) A `VeryGood` type object cannot be assigned to a variable of type `Good`.
(B) The `Good` class has a constructor but does not have a no-argument constructor.
(C) The `VeryGood` class does not have any explicitly defined constructors.
(D) The compiler cannot decide which `doSomething` method to call when `x` is initialized.
(E) The `VeryGood` class has two `doSomething` methods with different signatures, and one calls the other.

33. The method `getPrimeFactors` below returns a list of all prime factors of n.

```
public ArrayList<Integer> getPrimeFactors(int n)
{
  ArrayList<Integer> factors = new ArrayList<Integer>();

  for (int p = 2; p <= n; p++)
  {
    while (n % p == 0)
    {
      factors.add(p);
      n /= p;
    }
  }
  return factors;
}
```

How many times the modulo division operator `%` is invoked when `getPrimeFactors(60)` is called?

(A) 3
(B) 4
(C) 7
(D) 8
(E) 15

34. Consider the following method `sort`.

```
public static void sort(int[] a)
{
  for (int i = 1; i < a.length; i++)
  {
    int temp = a[i];
    int j = i-1;
    while (j >= 0 && a[j] > temp)
    {
      a[j+1] = a[j];
      j--;
    }
    a[j+1] = temp;
  }
}
```

Which sorting algorithm is implemented in this method?

(A) Insertion Sort
(B) Selection Sort
(C) Mergesort
(D) A hybrid algorithm that combines Selection Sort and Insertion Sort
(E) A hybrid algorithm that combines Insertion Sort and Mergesort

35. In the "composite" design pattern, a class extends a base class and also has a list of objects of the base class. For example,

```
public class Folder extends File
{
  private ArrayList<File> files;

  public Folder(String name)
  { < code not shown > }

  public boolean add(File f)
  {
    if ( < condition > )
      return files.add(f);
    else
      return false;
  }
}
```

`Folder`'s constructor passes the parameter `name` to its superclass's constructor and initializes the `ArrayList` `files` to an empty list.

The `equals` method in `File` compares the names of files:

```
public class File
{
  public boolean equals(File other)
  {
    return getName().equals(other.getName());
  }

  < instance variables, constructors, and other methods not shown >
}
```

The `Folder` class inherits the `equals` method from `File`. Note that a folder and a file inside it can have the same name.

We want to disallow adding a `null` or the folder itself to its `files` list. Which of the following expressions can replace < *condition* > in `Folder`'s `add` method to accomplish this?

I. `f != this && f != null`

II. `f != null && !f.equals(this)`

III. `!f.equals(null) && !f.equals(this)`

(A) I only
(B) II only
(C) I and II only
(D) II and III only
(E) I, II, and III

36. What is printed when the following code segment is compiled and executed?

```
String s = "ABC";
for (int i = 0; i < s.length(); i++)
  s += s.substring(i);
System.out.println(s);
```

(A) ABCABC
(B) ABCAABABC
(C) ABCABCBCC
(D) StringIndexOutOfBoundsException
(E) Memory overflow error

37. What is the output when the following code segment is compiled and executed?

```
int[] arr = {-3, 0, 2, -5, 6, 4};

for (int i = 1; i < arr.length; i++)
{
  if (arr[i] > 0 && arr[0] < 0)
  {
    int temp = arr[0];
    arr[0] = arr[i];
    arr[i] = temp;
  }
}

for (int x : arr)
{
  System.out.print(x + " ");
}
```

(A) -3 0 2 -5 6 4
(B) 6 0 2 -5 -3 4
(C) 4 0 2 -5 6 -3
(D) 2 6 4 -3 0 -5
(E) 2 0 -3 -5 6 4

38. Given

```
double[][] mat = { < some values > };
double sum = 0;
```

which of the following code segments DOES NOT assign to `sum` the sum of all the elements of `mat`?

(A)

```
for (int i = 0; i < mat.length; i++)
{
  for (int j = 0; j < mat[i].length; j++)
  {
    sum += mat[i][j];
  }
}
```

(B)

```
for (double x : mat)
{
  sum += x;
}
```

(C)

```
for (double[] row : mat)
{
  for (double x : row)
  {
    sum += x;
  }
}
```

(D)

```
for (int i = 0; i < mat.length; i++)
{
  for (double x : mat[i])
  {
    sum += x;
  }
}
```

(E)

```
for (double[] row : mat)
{
  for (int j = 0; j < row.length; j++)
  {
    sum += row[j];
  }
}
```

39. What is printed when the following code segment is executed?

```
ArrayList<String> words = new ArrayList<String>();
words.add("One");
words.add("Two");
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(1);
numbers.add(2);
ArrayList<Object> mix = new ArrayList<Object>();
mix.add(words);
mix.add("and");
mix.add(numbers);
System.out.println(mix);
```

(A) `[and]`
(B) `[ArrayList, and, ArrayList]`
(C) `[One, Two, and, 1, 2]`
(D) `[[One, Two], and, [1, 2]]`
(E) `[[One, Two], [and], [1, 2]]`

40. Given a class `B` and its subclass `D`, how many among the following five statements are true?

- If no constructors are defined in `D`, Java provides a default no-argument constructor that implicitly invokes the no-argument constructor of `B`.
- A constructor in `D` can explicitly invoke a constructor in `B` by using the keyword `super`; the statement with the keyword `super` must be the first statement in `D`'s constructor.
- A method in `D` can call a method of `B` using the keyword `super` with a dot and the method's name and appropriate parameters.
- A private method of `D` can override a public method of `B` with the same signature.
- A method in `D` can override a method of `B`; `D` can also include an overloaded method with the same name but a different signature.

(A) 1
(B) 2
(C) 3
(D) 4
(E) 5

Practice Exam #1

SECTION II

Time — 1 hour and 30 minutes
Number of questions — 4
Percent of total grade — 50

1. The *totient* function $\varphi(n)$ is the count of numbers from 1 to n that are *relatively prime* with n, that is, have no common divisors with n except 1. For example, 1, 5, 7, and 11 are relatively prime with 12, while 2, 3, 4, 6, 8, 9, and 10 have a common divisor with 12, so $\varphi(12) = 4$. This function was introduced by Leonhard Euler (1707-1783), one of the greatest mathematicians of all times. It is used in number theory and in certain algorithms. The totient function is traditionally written using the Greek letter φ (phi).

   Euler also described a simple way to calculate $\varphi(n)$. As we know, any integer $n > 1$ can be represented as the product of distinct prime factors, raised to the appropriate powers. For example, $12 = 2 \cdot 2 \cdot 3 = 2^2 \cdot 3^1$ and $200 = 2 \cdot 2 \cdot 2 \cdot 5 \cdot 5 = 2^3 \cdot 5^2$. Euler proved that if $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$, then $\varphi(n) = (p_1 - 1)p_1^{e_1 - 1} \cdot (p_2 - 1)p_2^{e_2 - 1} \cdot \ldots \cdot (p_k - 1)p_k^{e_k - 1}$. For example, $\varphi(12) = (2-1) \cdot 2^1 \cdot (3-1) \cdot 3^0 = 4$ and $\varphi(200) = (2-1)2^2 \cdot (5-1) \cdot 5^1 = 80$.

   In this question you will write two methods that help calculate $\varphi(n)$ using Euler's formula.

   (a) Write a method `smallestPrimeFactor(int n)` that returns the smallest prime factor of $n \geq 2$. Note that the smallest prime factor of n is simply the smallest integer greater than or equal to 2 that divides n.

   Complete the method `smallestPrimeFactor` below.

```
/** Returns the smallest prime factor of n.
 *  Precondition: n >= 2.
 */
public static int smallestPrimeFactor(int n)
```

(b) Write a method `totient` that returns $\varphi(n)$. By convention, $\varphi(1) = 1$. For $n \geq 2$, you must use Euler's formula $\varphi(n) = (p_1 - 1)p_1^{e_1 - 1} \cdot (p_2 - 1)p_2^{e_2 - 1} \cdot \ldots \cdot (p_k - 1)p_k^{e_k - 1}$ for $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$. Your solution will not receive full credit if you count the number of integers below n that are relatively prime with n by "brute force." Call the `smallestPrimeFactor` method from Part (a). Assume that it works correctly, regardless of what you wrote in Part (a). Solutions that include code that can be replaced by a call to `smallestPrimeFactor` will not receive full credit.

Complete the method `totient` below.

```
/** Returns the value of Euler's totient function of n.
 *  Precondition: n >= 1.
 */
public static int totient(int n)
```

For additional practice, write code to find the integer n from 1 to 1000 for which $\frac{\varphi(n)}{n}$ has the smallest value.

2. This question involves an implementation of the class `ReviewedProduct` that tracks reviews for a product. A `ReviewedProduct` object is created with one parameter, the product's name (a `String`). The `ReviewedProduct` class provides one constructor and the following four methods:

   - `addReview`, a `void` method that takes one integer parameter, the number of stars, and updates the relevant instance variables. The number of stars passed to `addReview` is always from 1 to 5; this can be assumed to be a precondition.

   - `avgStars`, which returns a `double` value. It returns the average number of stars for all reviews, rounded to the nearest half-star. `avgStars` returns `0.0` if no reviews have been entered for this product yet.

   - `isBad`, which returns a `boolean` value. `isBad` returns `true` if the product has been reviewed at least three times and at least half of the reviews were one-star reviews; otherwise `isBad` returns `false`.

   - `toString`, which overrides `Object`'s `toString` method and returns the name of this product followed by the average number of stars and the word "stars".

   The following table contains a sample code execution sequence and the corresponding results.

Statements and Expressions	Value Returned (blank if no value)	Comment
`ReviewedProduct pr = new ReviewedProduct("Widget");`		A new product `pr` is created.
`pr.avgStars();`	`0.0`	No reviews have been added yet.
`pr.addReview(1);`		A one-star review added.
`pr.isBad();`	`false`	
`pr.addReview(1);`		A one-star review added.
`pr.isBad();`	`false`	
`pr.addReview(5);`		A five-star review added.
`pr.isBad();`	`true`	
`pr.addReview(2);`		A two-star review added.
`pr.isBad();`	`true`	
`pr.avgStars();`	`2.5`	The average for all reviews added so far, rounded (up) to half-star.
`pr.addReview(1);`		A one-star review added.
`pr.isBad();`	`true`	
`pr.avgStars();`	`2.0`	The average for all reviews, 10/5, rounded to the nearest one-half.
`System.out.println(pr);`		Displays `Widget 2.0 stars`

Write the complete `ReviewedProduct` class, including the constructor and any required instance variables and methods. Do not use any arrays or lists. Your implementation must meet the above specifications, conform to the example, and be consistent with the information hiding principle.

For additional practice, make the `ReviewedProduct` class extend a small class `Product`, move the instance variable that holds the name of the product to `Product`, make `ReviewedProduct`'s constructor invoke `Product`'s constructor and `ReviewedProduct`'s `toString` method override and call `Product`'s `toString` method.

3. The *Set* game is played with a deck of 81 cards. Each card has one, two, or three identical shapes and four attributes: number of shapes, kind of shape (diamond, oval, or squiggle), color (red, green, or purple), and fill (solid, striped, or none). The cards in the deck have all possible combinations of attributes, hence 3*3*3*3 = 81 cards.

   At the beginning of the game, 12 cards are placed face-up on the table, while the rest remain face down in the deck. The players try to find a "set" of three cards among the open cards. The definition of what constitutes a "set" is not important in answering this question (it will be explained below, in the optional lab based on this question). When a player finds a set, the three cards that form the set are removed from the table (given to the player who found the set) and three cards from the deck are opened and added to the table. If none of the players can find a set, three cards from the deck are opened and added to the table.

   In a computer implementation of the game, a card is represented by an object of the class `Card`. This class defines the following static method.

```
/** Returns true if card1, card2, and card3 form a set;
 *  otherwise returns false.
 */
public static boolean isSet(Card card1, Card card2, Card card3)
```

   The game is represented by an object of the class `SetGame`. You will write two methods of this class. A partial definition of the `SetGame` class is shown below.

```
public class SetGame
{
  /** The array that holds the deck */
  private Card[] deck;

  /** The number of cards left in the deck */
  private int deckSize;

  /** The list of open cards on the table */
  private ArrayList<Card> openCards;

  /** Constructor: Creates a deck of 81 cards,
   *  shuffles the deck, then removes twelve cards
   *  from the top of the deck (end of the array)
   *  and adds them to the openCards list.
   */
  public SetGame()
  { /* implementation not shown */ }

  /** Examines the openCards list.
   *  If a set is found, returns an int array of three
   *  elements, the indices of the three cards that form the set.
   *  If no set is found, returns null.
   *  Postcondition: if indices is the returned array,
   *                 indices[0] < indices[1] < indices[2]
   */
  public int[] findSet()
  { /* to be implemented in Part (a) */ }

  /** If the parameter is not null, removes the
   *  cards with the specified indices from the
   *  openCards list. If the deck is not empty,
   *  adds three cards from the deck to the openCards
   *  list and decrements the deck size accordingly.
   *  Precondition: indices is null or
   *                indices[0] < indices[1] < indices[2]
   */
  public void updateTable(int[] indices)
  { /* to be implemented in Part (b) */ }

  /* other methods and instance variables are not shown */
}
```

(a) Write the `findSet` method of the `SetGame` class. The method examines all triplets of cards from the `openCards` list. If a "set" of three cards is found, as determined by the `isSet` method of the `Card` class, `findSet` returns an array of their indices in the `openCards` list. The returned array must be in ascending order. If no "set" is found, the method returns `null`.

Complete the `findSet` method below.

```
/** Examines the openCards list.
 *  If a set is found, returns an int array of three
 *  elements, the indices of the three cards that form the set.
 *  If no set is found, returns null.
 *  Postcondition: if indices is the returned array,
 *                 indices[0] < indices[1] < indices[2]
 */
public int[] findSet()
```

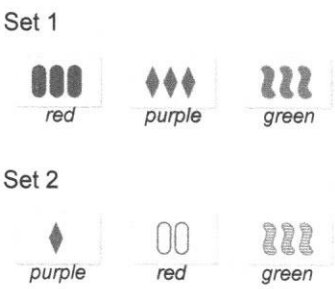
(b) Write the `updateTable` method of the `SetGame` class. The method removes the three cards with the specified indices from the `openCards` list and adds three cards from the deck, (or as many as are available, up to three) to the `openCards` list. The opened cards must be taken from the "top" of the deck, which is the last available card. The deck size is adjusted accordingly.

Complete the `updateTable` method below.

```
/** If the parameter is not null, removes the
 *  cards with the specified indices from the
 *  openCards list. If the deck is not empty,
 *  adds three cards from the deck to the openCards
 *  list and decrements the deck size accordingly.
 *  Precondition: indices is null or
 *                indices[0] < indices[1] < indices[2]
 */
public void updateTable(int[] indices)
```

For additional practice, write the `Card` class and the missing code in the `SetGame` class.

Three cards form a "set" if, for each of the four attributes, the values of that attribute for the three cards are either all the same or all different. The figure below shows two examples of "sets."



In the first "set," the numbers and fills are the same but the shapes and colors are all different. In the second set, all four attributes are different. Notice that three cards form a "set" if and only if for each attribute the sum of its values in the three cards is evenly divisible by 3.

Write the `Card` class. Provide a constructor that takes the values for the four attributes of the card (four integers with values 1, 2, or 3). It is convenient to store the attributes in an instance variable that is an array of four integers. Provide a static method `isSet`. The method takes three cards as parameters.

Write a constructor of the `SetGame` class. It should create a deck of 81 cards with all possible combinations of attributes, shuffle the deck, then "open" 12 cards, removing them from the top of the deck (the last available element in the `deck` array) and adding them to the `openCards` list.

Write a private `void` method `shuffle()` of the `SetGame` class to shuffle the deck. The shuffling algorithm can be similar to Selection Sort: set n to `deckSize`; while $n \geq 2$ choose a random card among the first n, swap it with the n-th card, decrement n.

4. The *Sliding Tiles* puzzle consists of a rectangular grid with marked square cells. A cell can be empty or hold a tile with a letter on it. Figure 1 shows an example.

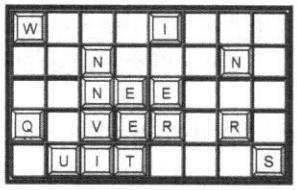


Figure 1

A player can slide a tile to a neighboring empty cell, up, down, left, or right. Let us call a configuration of tiles "packed" if no tile has a neighboring empty cell above or to the left of it. Figure 2 shows an example of a packed grid.

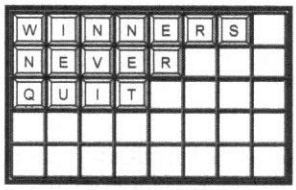


Figure 2

In a computer model of the game, the grid of cells is represented by a 2D array. A tile is represented by a one-letter literal `String` object; an empty cell holds a `null`.

(a) Write a static `boolean` method `isPacked` that returns `true` if a given configuration of tiles is "packed"; otherwise it returns `false`.

Complete the `isPacked` method below.

```
/** Returns true if the configuration of tiles represented
 *  by the 2D array tiles is packed, that is, any element
 *  that is not null does not have a null in the same
 *  column in the previous row or in the previous column
 *  in the same row; otherwise returns false.
 *  Precondition: tiles is not empty.
 */
public static boolean isPacked(String[][] tiles)
```

(b) To pack a given configuration of tiles, we can scan the whole grid row by row, starting at the top row, and each row from left to right. When we encounter a tile, we move the tile by sliding it up to the neighboring empty cell as far as possible, then to the left to the neighboring empty cell as far as possible. Write a private helper method `moveTile` that helps to implement this algorithm.

Complete the `moveTile` method below.

```
/** Relocates the tile at position (row, col) by first
 *  "sliding" it up several times to the previous row
 *  in the same column, as long as the cell there is empty;
 *  then sliding the tile to the left several times to
 *  the previous column in the same row, as long as the
 *  cell there is empty.
 *  Precondition: The 2D array tiles is not empty; the cell
 *                at location (row, col) holds a tile.
 */
private static void moveTile(String[][] tiles,
                                          int row, int col)
```

For additional practice, write a `void` method `pack(String[][] tiles)` that packs a given grid. Follow the algorithm described in Part (b) and use the `moveTile` method.

Write a static `void` method `display(String[][] tiles)` that prints out the grid, replacing `null`s with dots.

Declare a grid that matches Figure 1 above, print it out, pack it, and print out the resulting grid. The output should match the grid in Figure 2.

To test your code further, repeat the following 100 times: create an 8-by-5 grid filled with random letters and several `null`s in random places, then pack it and verify that it is indeed packed.