1. Consider the following code segment.

```
int x = 20;
int y = 6;

if ((x > 15) && (y > 10))
        System.out.println("Result: " + x * y);
else
        System.out.println("Result: " + x / y);
```

What is printed as a result of running the code segment?

A. Result: 20 * 6
B. Result: 20 / 6
C. Result: 120
D. Result: 3.3333333
E. Result: 3

2. Consider the following declarations:

```
String s1 = "new year";
String s2 = new String("new year");
String s3 = s1;
```

Which expression(s) involving these Strings evaluates to true?

I. **s1 == s2** II. **s1.equals(s2)** III. **s3.equals(s2)**

A. I only
B. II only
C. II and III only
D. I and II only
E. I, II, and III

3. Consider the following method.

```
public ArrayList <Integer> workonList (int num)
{
        ArrayList <Integer> sequence = new ArrayList <Integer> ();

        for (int i = 1; i <= num; i++)
                sequence.add(new Integer(i * i + 2));

        return sequence;
}
```

What is printed as a result of the following statement?

```
System.out.println(workonList(6));
```

A. [3, 8, 15, 24, 35]
B. [3, 8, 15, 24, 35, 48]
C. [0, 3, 6, 11, 18, 27]
D. [3, 6, 11, 18, 27]
E. [3, 6, 11, 18, 27, 38]

4. The statement **super( );** does which of the following?
   A. calls the method super as defined in the current class
   B. calls the method super as defined in the current class' parent class
   C. calls the method super as defined in java.lang
   D. calls the constructor as defined in the current class
   E. calls the constructor as defined in the current class' parent class

5. Consider the following method.

```
public String makeString (String input)
{
        String output = new String("");

        for (int k = 0; k < input.length(); k = k + 2)
        {
                output += input.substring(k, k + 1);
        }
        return output;
}
```

What is returned as a result of the call to **makeString("CUPERTINO")** ?

   A. **"CUPERTINO"**
   B. **"CPRIO"**
   C. **"UETN"**
   D. **"CUPERTIN"**
   E. Nothing is returned because an IndexOutOfBoundsException is thrown

6. Consider the following incomplete method that is intended to return an array that contains the contents of its first array parameter followed by the contents of its second array parameter.

```
public static int [] append (int [] a1, int [] a2)
{
        int [] result = new int [a1.length + a2.length];

        for (int j = 0; j < a1.length; j++)
                result[j] = a1[j];

        for (int k = 0; k < a2.length; k++)
                result[  /*  index  */  ] = a2[k];

        return result;
}
```

Which of the following expressions can be used to replace **/* index */** so that **append** will work as intended?

   A. **j**
   B. **k**
   C. **k + a1.length - 1**
   D. **k + a1.length**
   E. **k + a1.length + 1**

7. Consider the following method:

```
public String changeit (String message, String place)
{
        int i = message.indexOf(place);
        while (i >= 0)
        {
                message = message.substring(0, i) + " " + message.substring(i + place.length());
                i = message.indexOf(place);
        }
        return message;
}
```

What is the output of the following code segment?

```
String words = "I am\nso ready\nfor my\nFinal Exam(s)";
String words2 = changeit(words,"\n");
System.out.println(words2 + "\n" + words);
```

A.
I a so read for m Final Exam(s
I am
so ready
for my
Final Exam(s)

B.
I am so ready for my
I am
so ready
for my

C.
I am o ready or my inal Exam(s)
I am
so ready
for my
Final Exam(s)

D.
I am so ready for my Final Exam(s)
I am
so ready
for my
Final Exam(s)

E.
I am
so ready
for my
Final Exam(s)
I am
so ready
for my
Final Exam(s)

8. Consider the following method, **isSorted**, which is intended to return **true** if an array of integers is sorted in nondecreasing order and to return **false** otherwise.

```
public static boolean isSorted (int [] data)
{
        /*  missing code  */
}
```

Which of the following can be used to replace **/* missing code */** so that **isSorted** will work as intended?

I.
```
for (int k = 1; k < data.length; k++)
{
        if (data[k - 1] > data[k])
                return false;
}
return true;
```

II.
```
for (int k = 0; k < data.length; k++)
{
        if (data[k] > data[k + 1])
                return false;
}
return true;
```

III.
```
for (int k = 0; k < data.length - 1; k++)
{
        if (data[k] > data[k + 1])
                return false;
        else
                return true;
}
return true;
```

A. I only
B. II only
C. III only
D. I and II only
E. I and III only

9. What is the output of the following code segment?

```
String mv1 = "I Love ";
String mv2 = mv1;
mv2 += "Cupertino! ";
mv2.substring(7);
System.out.println(mv1 + mv2);
```

A. **I Love Cupertino!**
B. **I Love I Love Cupertino!**
C. **I Love Cupertino! Cupertino!**
D. **I Love Cupertino! I Love Cupertino!**
E. **I Love Cupertino! I Love**

For questions 10 – 12, use the following partial class definitions:

```
public class Teacher
{
        public String [] studentNames;
        private int [][] testScores;
        protected char [] letterGrades;
        …
}

public class CompSciTeacher extends Teacher
{
        protected int [][] programScores;
        private int [] seatNumbers;
        …
}

public class APCompSciTeacher extends CompSciTeacher
{
        private int [] apScores;
        …
}
```

10. Which of the following lists of instance data are accessible in class **CompSciTeacher**?
    A. studentNames, testScores, letterGrades, programScores, seatNumbers
    B. studentNames, testScores, letterGrades, programScores
    C. studentNames, letterGrades, programScores, seatNumbers
    D. letterGrades, programScores, seatNumbers
    E. programScores, seatNumbers

11. Which of the following lists of instance data are accessible in class **APCompSciTeacher**?
    A. studentNames, testScores, letterGrades, programScores, seatNumbers, apScores
    B. programScores, seatNumbers, apScores
    C. programScores, apScores
    D. studentNames, letterGrades, programScores, apScores
    E. studentNames, programScores, apScores

12. Which of the following is true regarding the use of instance data **testScores** of class **Teacher**?
    A. it is accessible in **Teacher**, **CompSciTeacher**, and **APCompSciTeacher**
    B. it is accessible in **Teacher** and **CompSciTeacher**
    C. it is accessible in **Teacher** only
    D. it is accessible in **CompSciTeacher** only
    E. it is accessible in **APCompSciTeacher** only

13. The expression **!((a <= b) && (c > 3))** is equivalent to which of the following expressions?

    A. **((a <= b) && (c > 3))**
    B. **((a <= b) || (c > 3))**
    C. **((a > b) || (c > 3))**
    D. **((a > b) || (c <= 3))**
    E. **((a > b) && (c <= 3))**

14. All classes in Java are directly or indirectly subclasses of the _____ class.

A Wrapper
B. String
C. Reference
D. this
E. Object

15. Consider the following code segment.

```
int [] array = {9, 10, 5, 8, 12, 3};

for (int k = 0; k < array.length - 1; k++)
{
        if (array[k] < array[k + 1])
                System.out.print(k + "  " + array[k] + "  ");
}
```

What will be printed as a result of executing the code segment?

A. 1 10 5 12
B. 0 9 2 5 3 8
C. 0 9 2 5 4 12
D. 1 10 3 8 5 3
E. 0 9 3 8 5 3

16. Suppose an interface **Solid** specifies the **getSurfaceArea()** method. Two classes, **Sphere** and **Cube**, implement **Solid**. Which Java feature makes it possible for the following code segment to print the correct values for the surface area of a sphere and a cube?

```
Solid[] solids = new Solid[2];
solids[0] = new Sphere(12);
solids[1] = new Cube(31);
System.out.println("Sphere: " + solids[0].getSurfaceArea());
System.out.println("Cube: " + solids[1].getSurfaceArea());
```

A. abstraction
B. encapsulation
C. polymorphism
D. platform independence
E. method overloading

17. Which of the following is (are) true?

I. An abstract class must contain an abstract method.
II. An abstract class can contain a constructor.
III. An abstract class can contain public fields (data members).

A. I only
B. I and II only
C. I and III only
D. II and III only
E. I, II, and III

18. The class **CourseList** provides methods that allow you to represent and manipulate a list of high school courses, but you are not concerned with how these operations work or how the list is stored in memory. You only know how to initialize and use **CourseList** objects and have no direct access to the implementation of the **CourseList** class or its private data fields. This is an example of

A. method overloading
B. polymorphism
C. inheritance
D. overriding
E. encapsulation

19. Consider the following method:

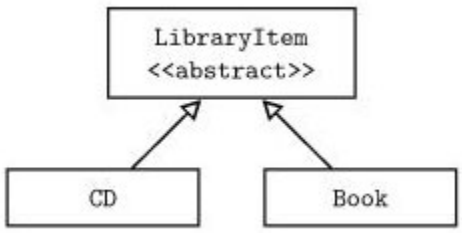
```
public void change (double [] nums, int n)
{
        for (int k = 0; k < n; k++)
        {
                nums[k] = 5.4;
        }
        n = 2;
}
```

What will be stored in **samples** and **len** after the following statements are executed?

```
double [] samples = {1.0, 2.1, 3.2, 4.3};
int len = samples.length;
change(samples,len);
```

A. **samples** contains **5.4**, **5.4**, **5.4**, **5.4** and **len** is **4**
B. **samples** contains **5.4**, **5.4**, **5.4**, **5.4** and **len** is **2**
C. **samples** contains **1.0**, **2.1**, **3.2**, **4.3** and **len** is **4**
D. **samples** contains **5.4**, **5.4** and **len** is **2**
E. **samples** contains **1.0**, **2.1** and **len** is **2**

20. Consider the following inheritance hierarchy diagram:



Which of the following declarations will **not** cause an error? You may assume that each of the classes has a default constructor.

I. **LibraryItem item = new LibraryItem();**
II. **Book valedor = new LibraryItem();**
III. **LibraryItem mariah = new CD();**

A. I only
B. II only
C. III only
D. II and III only
E. I, II, and III

Questions 21 and 22 refer to the following information.

Consider the following partial class declaration.

```
public class ShowSomeClass
{
        private int myValue;        private int myNumber;        private int myInteger;

        //  Constructors not shown

        public int getmyValue ( )
        {       return myValue;       }

        public void setmyNumber (int num)
        {       myNumber = num;       }
}
```

21. The following declaration appears in another class.

```
ShowSomeClass classy = new ShowSomeClass();
```

Which of the following will compile without error?

A. **int x = classy.getmyValue();**

B. **int x;**
classy.getmyValue(x);

C. **int x = classy.myValue;**

D. **int x = ShowSomeClass.getmyValue();**

E. **int x = getmyValue(classy);**

22. Which of the following changes to **ShowSomeClass** will allow other classes to access but not modify the value of **myInteger**?

A. make **myInteger** public.

B. Include the method:
```
public int getmyInteger ( )
{       return myInteger;       }
```

C. Include the method:
```
private int getmyInteger ( )
{       return myInteger;       }
```

D. Include the method:
```
public void getmyInteger (int x)
{       x = myInteger;       }
```

E. Include the method:
```
private void getmyInteger (int x)
{       x = myInteger;       }
```

23. Consider the following class declaration.

```
public class Student
{
        private String myName;
        private int myAge;

        public Student ( )
        {       /*  implementation not shown  */       }

        public Student (String name, int age)
        {       /*  implementation not shown  */       }
}
```

Which of the following declarations will compile without error?

I. **Student genius = new Student();**

II. **Student smartypants = new Student("Sally", 15);**

III. **Student brainy = new Student("Suzy", "15");**

A. I only
B. II only
C. I and II only
D. I and III only
E. I, II, and III

24. Consider the following method.

```
public static int arrayop (int [] arr)
{
        int x = 0;

        for (int k = 0; k <= arr.length; k = k + 2)
        {
                x = x + arr[k];
        }
        return x;
}
```

Assume that the array nums has been declared and initialized as follows:

```
int [] nums = {3, 5, 1, 0, 4, 6, 7, 2, 8};
```

What value will be returned as a result of the call **arrayop(nums)** ?

A. 13
B. 15
C. 23
D. 36
E. Array Index Out Of Bounds Exception

25. Consider the following method that is intended to determine if the **double** values **num1** and **num2** are close enough to be considered equal. For example, given a **tolerance** of **0.001**, the values **23.32271** and **23.32294** would be considered equal.

```
// return true if num1 and num2 are within the specified tolerance, false otherwise
public boolean almostEqual (double num1, double num2, double tolerance)
{
        /* missing code */
}
```

Which of the following should replace **/* missing code */** so that **almostEqual** will work as intended?

A. **return (num1 – num2) <= tolerance;**
B. **return ((num1 – num2) / 2) <= tolerance;**
C. **return (num1 – num2) >= tolerance;**
D. **return ((num1 – num2) / 2) >= tolerance;**
E. **return Math.abs(num1 – num2) <= tolerance;**

26. Consider the following interface and class declarations.

```
public interface Vehicle
{
        double getMileage();                    // returns the mileage traveled by this Vehicle
}

public class Fleet
{
        private ArrayList <Vehicle> myVehicles;

        public double getTotalMileage ( )       // returns the mileage traveled by all vehicles in this Fleet
        {
                double sum = 0.0;

                for (Vehicle v : myVehicles)    // this for-each loop traverses the Vehicles in the ArrayList
                {
                        sum += /* expression */ ;
                }

                return sum;
        }

        // there may be instance variables, constructors, and methods not shown
}
```

Which of the following can be used to replace **/* expression */** so that **getTotalMileage** returns the total of the miles traveled for all vehicles in the fleet?

A. **getMileage(v)**
B. **myVehicles[v].getMileage()**
C. **Vehicle.get(v).getMileage()**
D. **myVehicles.get(v).getMileage()**
E. **v.getMileage()**

27. Consider the following class declaration.

```
public class Person
{
        private String myName;
        private String faveSchool;

        public Person (String name, String school)
        {
                myName = name;
                faveSchool = school;
        }

        public String getSchool ( )
        {       return faveSchool;      }

        public void setSchool (String school)
        {       faveSchool = school;    }
}
```

Assume that the following declaration has been made.

```
Person smartstudent = new Person("Rajeev", "Los Gatos");
```

Which of the following statements is the most appropriate for changing the name of faveSchool from “Los Gatos” to “Monta Vista”?

A. **smartstudent = new Person("Rajeev", "Monta Vista");**

B. **smartstudent.faveSchool = "Monta Vista";**

C. **smartstudent.getSchool("Monta Vista");**

D. **smartstudent.setSchool("Monta Vista");**

E. **Person.setSchool("Monta Vista");**

28. Consider the following code segment

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

for (int k = 3; k < array.length – 1; k++)
        array[k] = array[k + 1];
```

Which of the following represents the contents of **array** as a result of executing the code segment?

A. **{1, 2, 3, 4, 5, 6, 7}**
B. **{1, 2, 3, 5, 6, 7}**
C. **{1, 2, 3, 5, 6, 7, 7}**
D. **{1, 2, 3, 5, 6, 7, 8}**
E. **{2, 3, 4, 5, 6, 7, 7}**

29. Consider the following method:

```
public void schoolpride(int n)
{
        if (n <= 0)
                return;

        schoolpride (n-1);

        for(int i = 0; i < n; i++)
        {
                System.out.print("M");
        }
        for(int i = 0; i < n; i++)
        {
                System.out.print("V");
        }
        System.out.println();
}
```

What is the output when **schoolpride(4)** is called?

A.
```
MMMMVVVV
```

B.
```
MMMMVVVV
MMMMVVVV
MMMMVVVV
MMMMVVVV
```

C.
```
MMMMV
MMMMVV
MMMMVVV
MMMMVVVV
```

D.
```
MV
MMVV
MMMVVV
MMMMVVVV
```

E.
```
MMMMVVVV
MMMVVV
MMVV
MV
```

30. Assume that **myList** is an **ArrayList** that has been correctly constructed and populated with objects. Which of the following expressions produces a valid random index for **myList**?

A. **(int)(Math.random() * myList.size()) - 1**
B. **(int)(Math.random() * myList.size())**
C. **(int)(Math.random() * myList.size()) + 1**
D. **(int)(Math.random() * (myList.size() + 1))**
E. **Math.random(myList.size())**

31. What is the output of the following code segment?

```
ArrayList <String> list = new ArrayList <String> ();
list.add("A");
list.add("B");
list.add("C");
list.add("D");
list.add("E");

for (int k = 1; k <= 3; k++)
{
        list.remove(1);                 //  notice that the argument is 1 here
}

for (int k = 1; k <= 3; k++)
{
        list.add(1,"*");                //  notice that the first argument is 1 here
}

for (String word : list)                //  this is a for-each loop; it will access the elements of the
{                                       //  ArrayList in order
        System.out.println (word + " ");
}
```

A. **A C D E * * ***
B. *** * * B C D E**
C. **A * * * E**
D. **A E * * ***
E. *** * * A E**

32. Consider the following method.

```
public static void actOnArray (int [] nums)
{
        int j = 0;
        int k = nums.length – 1;

        while (j < k)
        {
                int x = nums[j];
                nums[j] = nums[k];
                nums[k] = x;
                j++;
                k--;
        }
}
```

Which of the following describes what the method **actOnArray** does to the array **nums**?

A. The array **nums** is unchanged.
B. The first value in **nums** is copied to every location in the array.
C. The last value in **nums** is copied to every location in the array.
D. The method generates an **ArrayIndexOutOfBoundsException**.
E. The contents of the array **nums** are reversed.

33. Why doesn't Java let you create an object of the **Math** class?
    A. Because **Math** has no fields
    B. Because **Math** has no constructors
    C. Because all **Math**'s methods and fields are static, so all **Math** objects would be identical
    D. Because **Math** does not represent a real-world object
    E. Because **Math**'s coolness will not let it be objectified

34. Which of the following messages passed to the **String str** could throw a **StringIndexOutOfBoundsException**?
    A. **str.length()**
    B. **str.charAt(2)**
    C. **str.replace('a','A')**
    D. **str.equals(str)**
    E. any of the above could throw a **StringIndexOutOfBoundsException**

35. What will array **arr** contain after the following code segment has been executed?

```
int [] arr = {4, 3, 2, 1, 0};
for (int i = 1; i < arr.length; i++)
{
        arr[i-1] += arr[i];
}
```

    A. **10, 6, 3, 1, 0**
    B. **7, 5, 3, 1, 0**
    C. **7, 3, 2, 1, 0**
    D. **4, 7, 9, 10, 10**
    E. **4, 7, 5, 3, 1**

36. Consider the following method.

```
public static String scramble (String word, int howFar)
{
        return word.substring(howFar + 1, word.length()) + word.substring(0, howFar);
}
```

What value is returned as a result of the call **scramble("compiler", 3)** ?

    A. **"compiler"**
    B. **"pilercom"**
    C. **"ilercom"**
    D. **"ilercomp"**
    E. No value is returned because an **IndexOutOfBoundsException** will be thrown.

37. The relationship between a child (sub) class and a parent (super) class is referred to as a _____ relationship.
    A. has-a
    B. is-a
    C. was-a
    D. instance-of
    E. whatever-a

For problems 38 and 39, consider the following partial class definitions:

```
public abstract class MVStudent
{
        public MVStudent()  { . . . }
}

public class APStudent extends MVStudent
{
        private int yearinschool;
        public APStudent(int year)
        {
                super();
                yearinschool = year;
        }
}

public class APCompSciStudent extends APStudent
{
        private String accountname;
        public APCompSciStudent(String acct, int yr)
        {
                /*  missing statement(s) */
        }
}
```

38. Which of the following is an acceptable replacement for **/* missing statement(s) */** in **APCompSciStudent**'s constructor?

I. **yearinschool = yr;**
accountname = acct;

II. **super(yr);**
accountname = acct;

III. **super(acct, yr);**

A. I only
B. II only
C. I and II only
D. II and III only
E. I, II, and III

39. Which of the following declarations are valid for the partial class definitions above?

I. **MVStudent student = new APStudent(11);**

II. **APCompSciStudent student = new APStudent(11);**

III. **APStudent student = new APCompSciStudent("ss5005547",11);**

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

40. Consider the following classes:

```
public class Dog
{
        public Dog()
        {
                Bark();
        }
        public void Bark()
        {
                System.out.print("WOOF");
        }
}

public class Chihuahua extends Dog
{
        public Chihuahua ()
        {
                System.out.print("-");
        }
        public void Bark()
        {
                System.out.print("YIP");
        }
}
```

What is the output when the following code statement is executed?

```
Chihuahua fufu = new Chihuahua();
Dog fido = new Chihuahua();
Dog spike = new Dog();
```

A. **YIP-YIP-WOOF**
B. **YIP-WOOF-WOOF**
C. **-YIP-WOOF**
D. **--WOOF**
E. **WOOF-YIP-WOOF**

41. Consider the following code segment.

```
int sum = 0;
int k = 1;

while (sum < 12 || k < 4)
        sum += k;

System.out.println(sum);
```

What is printed as a result of executing the code segment?

A. 6
B. 10
C. 12
D. 15
E. Nothing is printed due to an infinite loop.

Questions 42 and 43 refer to the following information.

Consider the following instance variable and methods. You may assume that **testscores** has been initialized with length > 0. The methods are intended to return the index of an array element equal to **target**, or **-1** if no such element exists.

```
private int [] testscores;

public int searchIt (int target)
{
        return searchItHelper(target, testscores.length-1);
}

private int searchItHelper (int targ, int last)
{
        //  Line 1

        if (testscores[last] == targ)
                return last;
        else
                return searchItHelper(targ,last-1);
}
```

42. For which of the following test cases will **searchIt(94)** always result in an error?

I. **testscores** contains only one element.

II. **testscores** does not contain the value **94**.

III. **testscores** contains the value **94** multiple times.

A. I only
B. II only
C. III only
D. I and II only
E. I, II, and III

43. Which of the following should be used to replace **// Line 1** in **searchItHelper** so that **searchIt** will work as intended?

A.
```
if (last <= 0)
        return -1;
```

B.
```
if (last < 0)
        return -1;
```

C.
```
if (last < testscores.length)
        return -1;
```

D.
```
while (last < testscores.length)
```

E.
```
while (last >= 0)
```

44. Consider the following method.

```
public int compute(int n, int k)
{
        int answer = 1;

        for (int i = 1; i <= k; i++)
                answer *= n;

        return answer;
}
```

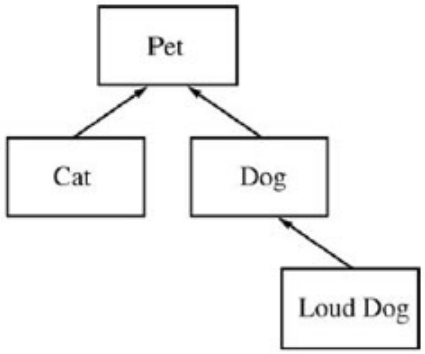
Which of the following represents the value returned as a result of the call **compute(n, k)** ?

A. n * k
B. n!
C. n to the k power
D. 2 to the k power
E. k to the n power

45. What is an "**abstract**" class?

A. A class that can be hard to interpret, like the modern art of Picasso
B. A class in which all fields and methods are static
C. A class where some methods are abstract
D. A class where all methods are abstract
E. Any class derived from the **Object** class

1. Consider the hierarchy of classes shown in the following diagram.



Note that a **Cat** "is-a" **Pet**, a **Dog** "is-a" **Pet**, and a **LoudDog** "is-a" **Dog**. The class **Pet** is shown in the following declaration.

```
public class Pet
{
        private String myName;

        public Pet (String name)
        {  myName = name;  }

        public String getName()
        {  return myName;  }

        public String speak()
        {  return "ah-OOOOO";  }
}
```

The subclass **Dog** has the partial class declaration shown below.

```
public class Dog extends Pet
{
        public Dog (String name)
        {  /* implementation not shown  */  }

        public String speak ()
        {  /* implementation not shown  */  }
}
```

A. Given the class hierarchy shown above, write a complete class declaration for the class **Cat**, including implementations of its constructor and method(s). The **Cat** method **speak** returns "meow" when it is invoked. Note that the method **speak** will need to be overridden in **Cat**.

B. Assume that class **Dog** has been declared as shown at the beginning of the question. If the String *dog-sound* is returned by the **Dog** method **speak**, then the **LoudDog** method **speak** returns a String containing *dog-sound* repeated two times. (note that we do not know the String for *dog-sound*)
Given the class hierarchy shown previously, write a complete class declaration for the class **LoudDog**, including implementations of its constructors and method(s).

2. Consider a class, **NoteKeeper**, that is designed to store and manipulate a list of short notes. Here are some typical notes:

pick up drycleaning
special dog chow
dog registration
dentist Monday
study for APCS quiz

The incomplete class declaration is shown below:

```
public class NoteKeeper
{
    private ArrayList<String> noteList;

    // Postcondition:  Prints all notes in noteList, one per line.
    //                 Notes are numbered  1, 2, 3, …
    //                 Each number is followed by a period and a space.
    public void printNotes ( )
    {
        /*  to be implemented in part (a)  */
    }

    // Postcondition:  All notes with specified word have been removed from noteList, leaving the order of the
    //                 remaining notes unchanged.  If none of the notes in noteList contains word, the list remains
    //                 unchanged.
    public void removeNotes(String word)
    {
        /*  to be implemented in part (b)  */
    }

    // Constructor and other methods not shown . . .
}
```

(a) Write the **NoteKeeper** method **printNotes**. The **printNotes** method prints all of the notes in **noteList**, one per line, and numbers the notes, starting at 1. The output should look like this:

1. pick up drycleaning
2. special dog chow
3. dog registration
4. dentist Monday
5. study for APCS quiz

Complete method **printNotes** below.

```
// Postcondition:  Prints all notes in noteList, one per line.
//                 Notes are numbered  1, 2, 3, …
//                 Each number is followed by a period and a space.
public void printNotes ( )
```

(b) Write the **NoteKeeper** method **removeNotes**. Method **removeNotes** removes all notes from **noteList** that contain the word specified by the parameter. The ordering of the remaining notes should be left unchanged. For example, suppose that a **NoteKeeper** variable, **notes**, has a **noteList** containing

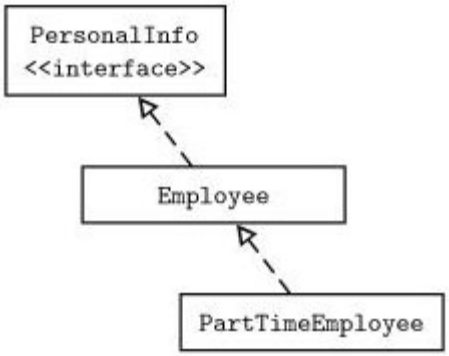
[pick up drycleaning, special dog chow, dog registration, dentist Monday, study for APCS quiz]

The method **notes.removeNotes("dog")** should modify the **noteList** of **notes** to be

[pick up drycleaning, dentist Monday, study for APCS quiz]

Complete method **removeNotes** below.

```
// Postcondition:  All notes with specified word have been removed from noteList, leaving the order of the
//                 remaining notes unchanged.  If none of the notes in noteList contains word, the list remains
//                 unchanged.
public void removeNotes(String word)
```



3. Consider the class hierarchy diagram shown to the right.

PersonalInfo is an interface that is implemented by **Employee**. A **PartTimeEmployee** is-a **Employee**. Here is the declaration for **PersonalInfo**.

```
public interface PersonalInfo
{
        String getName();
        String getCity();
        boolean getCitizenStatus();
}
```

(a) Given the class hierarchy diagram shown above, write a complete class declaration for the class **Employee**, including implementation of methods and a constructor with parameters. An **Employee**, in addition to implementing **PersonalInfo**, has a data field to store annual salary, and an accessor method that returns the annual salary of the **Employee**. Write the **Employee** class below, or on the multiple choice answer sheet.

(b) Write a complete class declaration for the **PartTimeEmployee** class, given the class hierarchy shown above. A **PartTimeEmployee** has two additional data fields:

- A real number that indicates the fraction that the part-time employee works (for example, 0.5, 0.8, etc.)
- A boolean field that stores whether the employee is a union member or not.

There are three additional methods:

- An accessor that returns the real number fraction that the **PartTimeEmployee** works.
- An accessor that returns a value indicating whether the **PartTimeEmployee** is a union member or not.
- A mutator that switches the status of that employee's union membership, but only if the employee's salary is greater than $15,000. In other words, if his/her salary is greater than $15,000, then the union membership status should be "flipped" from union to nonunion or vice versa.

Write the **PartTimeEmployee** class below, or on the multiple choice answer sheet.