Questions 4–5 refer to the Card and Deck classes shown below.

```
public class Card
{
    private String suit;
    private int value;      //0 to 12

    public Card(String cardSuit, int cardValue)
    { /* implementation */ }

    public String getSuit()
    { return suit; }

    public int getValue()
    { return value; }

    public String toString()
    {
        String faceValue = "";
        if (value == 11)
            faceValue = "J";
        else if (value == 12)
            faceValue = "Q";
        else if (value == 0)
            faceValue = "K";
        else if (value == 1)
            faceValue = "A";
        if (value >= 2 && value <= 10)
            return value + " of " + suit;
        else
            return faceValue + " of " + suit;
    }
}

public class Deck
{
    private Card[] deck;
    public final static int NUMCARDS = 52;

    public Deck()
    { ...

    /** Simulate shuffling the deck. */
    public void shuffle()
    { ...

    //Other methods are not shown.
}
```

OBJ & CLASS

#1

4. Which of the following represents correct /* *implementation* */ code for the constructor in the Card class?

(A)
```
suit = cardSuit;
value = cardValue;
```

(B)
```
cardSuit = suit;
cardValue = value;
```

(C)
```
Card = new Card(suit, value);
```

(D)
```
Card = new Card(cardSuit, cardValue);
```

(E)
```
suit = getSuit();
value = getValue();
```

5. Consider the implementation of a writeDeck method that is added to the Deck class.

```
/** Write the cards in deck, one per line. */
public void writeDeck()
{
    /* implementation code */
}
```

Which of the following is correct /* *implementation code* */?

I
```
System.out.println(deck);
```

II
```
for (Card card : deck)
    System.out.println(card);
```

III
```
for (Card card : deck)
    System.out.println((String) card);
```

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

19. Consider the `NegativeReal` class below, which defines a negative real number object.

```
public class NegativeReal
{
    private Double negReal;

    /** Constructor. Creates a NegativeReal object whose value is num.
     *  @param num  a negative real number
     */
    public NegativeReal(double num)
    { /* implementation not shown */ }

    /** @return the value of this NegativeReal */
    public double getValue()
    { /* implementation not shown */ }

    /** @return this NegativeReal rounded to the nearest integer */
    public int getRounded()
    { /* implementation */ }
}
```

Here are some rounding examples:

Negative real number	Rounded to nearest integer
−3.5	−4
−8.97	−9
−5.0	−5
−2.487	−2
−0.2	0

Which `/* implementation */` of `getRounded` produces the desired postcondition?

(A) `return (int) (getValue() - 0.5);`
(B) `return (int) (getValue() + 0.5);`
(C) `return (int) getValue();`
(D) `return (double) (getValue() - 0.5);`
(E) `return (double) getValue();`

22. Refer to the definitions of `ClassOne` and `ClassTwo` below.

```
public class ClassOne
{
    public void methodOne()
    {
        ...
    }

    //Other methods are not shown.
}

public class ClassTwo extends ClassOne
{
    public void methodTwo()
    {
        ...
    }

    //Other methods are not shown.
}
```

Consider the following declarations in a client class. You may assume that `ClassOne` and `ClassTwo` have default constructors.

```
ClassOne c1 = new ClassOne();
ClassOne c2 = new ClassTwo();
```

Which of the following method calls will cause an error?

I `c1.methodTwo();`

II `c2.methodTwo();`

III `c2.methodOne();`

(A) None
(B) I only
(C) II only
(D) III only
(E) I and II only

26. Refer to the following class, containing the `mystery` method.

```
public class SomeClass
{
    private int[] arr;

    /** Constructor. Initializes arr to contain nonnegative
     *  integers k such that 0 <= k <= 9.
     */
    public SomeClass()
    { /* implementation not shown */ }

    public int mystery()
    {
        int value = arr[0];
        for (int i = 1; i < arr.length; i++)
            value = value * 10 + arr[i];
        return value;
    }
}
```

Which best describes what the `mystery` method does?

(A) It sums the elements of `arr`.

(B) It sums the products `10*arr[0]` + `10*arr[1]` + ⋯ + `10*arr[arr.length-1]`.

(C) It builds an integer of the form $d_1 d_2 d_3 \ldots d_n$, where $d_1 =$ `arr[0]`, $d_2 =$ `arr[1]`, …, $d_n =$ `arr[arr.length-1]`.

(D) It builds an integer of the form $d_1 d_2 d_3 \ldots d_n$, where $d_1 =$ `arr[arr.length-1]`, $d_2 =$ `arr[arr.length-2]`, …, $d_n =$ `arr[0]`.

(E) It converts the elements of `arr` to base-10.

Use the following program description for Questions 32–34.

A programmer plans to write a program that simulates a small bingo game (no more than six players). Each player will have a bingo card with 20 numbers from 0 to 90 (no duplicates). Someone will call out numbers one at a time, and each player will cross out a number on his card as it is called. The first player with all the numbers crossed out is the winner. In the simulation, as the game is in progress, each player's card is displayed on the screen.

The programmer envisions a short driver class whose `main` method has just two statements:

```
BingoGame b = new BingoGame();
b.playBingo();
```

The `BingoGame` class will have several objects: a `Display`, a `Caller`, and a `PlayerGroup`. The `PlayerGroup` will have a list of `Players`, and each `Player` will have a `BingoCard`.

32. The relationship between the `PlayerGroup` and `Player` classes is an example of
    (A) an interface.
    (B) encapsulation.
    (C) composition.
    (D) inheritance.
    (E) independent classes.

33. Which is a reasonable data structure for a `BingoCard` object? Recall that there are 20 integers from 0 to 90 on a `BingoCard`, with no duplicates. There should also be mechanisms for crossing off numbers that are called, and for detecting a winning card (i.e., one where all the numbers have been crossed off).

I
```
int[] bingoCard;   //will contain 20 integers
                   //bingoCard[k] is crossed off by setting it to -1.
int numCrossedOff;   //player wins when numCrossedOff reaches 20.
```

II
```
boolean[] bingoCard;  //will contain 91 boolean values, of which
                      //20 are true. All the other values are false.
                      //Thus, if bingoCard[k] is true, then k is
                      //on the card, 0 <= k <= 90. A number k is
                      //crossed off by changing the value of
                      //bingoCard[k] to false.
int numCrossedOff;  //player wins when numCrossedOff reaches 20.
```

III
```
ArrayList<Integer> bingoCard;  //will contain 20 integers.
     //A number is crossed off by removing it from the ArrayList.
     //Player wins when bingoCard.size() == 0.
```

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

39. Consider the following two classes.

```
public class Bird
{
    public void act()
    {
        System.out.print("fly ");
        makeNoise();
    }

    public void makeNoise()
    {
        System.out.print("chirp ");
    }
}

public class Dove extends Bird
{
    public void act()
    {
        super.act();
        System.out.print("waddle ");
    }

    public void makeNoise()
    {
        super.makeNoise();
        System.out.print("coo ");
    }
}
```

Suppose the following declaration appears in a class other than `Bird` or `Dove`:

```
Bird pigeon = new Dove();
```

What is printed as a result of the call `pigeon.act()`?

(A) `fly`
(B) `fly chirp`
(C) `fly chirp waddle`
(D) `fly chirp waddle coo`
(E) `fly chirp coo waddle`

9. The `Tile` class contains a `toString` method that creates a `String` containing the letter and point value of a `Tile`. The string should be in the following format:

```
Letter letter (point value = pointValue)
```

For example,

```
Letter A (point value = 1)
Letter Z (point value = 10)
```

Consider the `toString` method below:

```
public String toString()
{
    return /* code */
}
```

Which `/* code */` leads to correct output?

(A) `Letter + "letter " + "(point value = " + pointValue + ")";`
(B) `"Letter " + letter + ("point value = " + pointValue);`
(C) `Letter + this.letter + " (point value = " + pointValue + ")";`
(D) `"Letter " + letter + " (point value = " + (String) pointValue + ")";`
(E) `"Letter " + letter + " (point value = " + pointValue + ")";`

10. Any two tiles in the word game that have the same letter also have the same point value, but the opposite is not necessarily true. For example, all the vowels have a point value of 1. Two tiles are said to match if they have the same letter. Consider the following `matches` method for the `Tile` class.

```
/** @return true if the letter on this tile equals the letter
 *  on otherTile */
public boolean matches(Tile otherTile)
{ return /* code */; }
```

Which replacements for `/* code */` return the desired result? Note: You may *not* assume that the `Tile` class has its own `equals` method.

I `letter == otherTile.letter`

II `this.equals(otherTile)`

III `letter.equals(otherTile.letter)`

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

Questions 2–4 refer to the `Worker` class below:

```
public class Worker
{
    private String name;
    private double hourlyWage;
    private boolean isUnionMember;

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

    public Worker(String aName, double anHourlyWage, boolean union)
    { /* implementation not shown */ }

    //Accessors getName, getHourlyWage, getUnionStatus are not shown.

    /** Permanently increase hourly wage by amt.
     *  @param amt the amount of wage increase
     */
    public void incrementWage(double amt)
    { /* implementation of incrementWage */ }

    /** Switch value of isUnionMember from true to false and
     *  vice versa.
     */
    public void changeUnionStatus()
    { /* implementation of changeUnionStatus */ }
}
```

2. Refer to the `incrementWage` method. Which of the following is a correct */* implementation of* `incrementWage` **/*?

   (A) `return hourlyWage + amt;`
   (B) `return getHourlyWage() + amt;`
   (C) `hourlyWage += amt;`
   (D) `getHourlyWage() += amt;`
   (E) `hourlyWage = amt;`

3. Consider the method changeUnionStatus. Which is a correct /* *implementation of* changeUnionStatus */?

I
```
if (isUnionMember)
    isUnionMember = false;
else
    isUnionMember = true;
```

II
```
isUnionMember = !isUnionMember;
```

III
```
if (isUnionMember)
    isUnionMember = !isUnionMember;
```

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

4. A client method computePay will return a worker's pay based on the number of hours worked.

```
/** Precondition: Worker w has worked the given number of hours.
 *  @param w a Worker
 *  @param hours the number of hours worked
 *  @return amount of pay for Worker w
 */
public static double computePay(Worker w, double hours)
{ /* code */ }
```

Which replacement for /* *code* */ is correct?

(A) `return hourlyWage * hours;`
(B) `return getHourlyWage() * hours;`
(C) `return w.getHourlyWage() * hours;`
(D) `return w.hourlyWage * hours;`
(E) `return w.getHourlyWage() * w.hours;`

ray to
ame as

9. An `Insect` class is to be written, containing the following data fields:
`age`, which will be initialized to `0` when an `Insect` is constructed.
`nextAvailableID`, which will be initialized to `0` outside the constructor and incremented each time an `Insect` is constructed.
`idNum`, which will be initialized to the current value of `nextAvailableID` when an `Insect` is constructed.
`position`, which will be initialized to the location in a garden where the `Insect` is placed when it is constructed.
`direction`, which will be initialized to the direction the `Insect` is facing when placed in the garden.

Which variable in the `Insect` class should be static?

(A) `age`
(B) `nextAvailableID`
(C) `idNum`
(D) `position`
(E) `direction`

T PAGE.

GO ON TO THE NEXT PAGE.

Questions 10 and 11 refer to the classes Address and Customer given below.

```
public class Address
{
    private String street;
    private String city;
    private String state;
    private int zipCode;

    public Address(String aStreet, String aCity, String aState,
            int aZipCode)
    { /* implementation not shown */ }

    public String getStreet()
    { /* implementation not shown */ }

    public String getCity()
    { /* implementation not shown */ }

    public String getState()
    { /* implementation not shown */ }

    public int getZipCode()
    { /* implementation not shown */ }

    //Other methods are not shown.
}

public class Customer
{
    private String name;
    private String phone;
    private Address address;
    private int ID;

    public Customer(String aName, String aPhone, Address anAddr,
            int anID)
    { /* implementation not shown */ }

    public Address getAddress()
    { /* implementation not shown */ }

    public String getName()
    { /* implementation not shown */ }

    public String getPhone()
    { /* implementation not shown */ }

    public int getID()
    { /* implementation not shown */ }

    //Other methods are not shown.
}
```

10. Which of the following correctly creates a Customer object c?

I
```
Address a = new Address("125 Bismark St", "Pleasantville",
    "NY", 14850);
Customer c = new Customer("Jack Spratt", "747-1674", a, 7008);
```

II
```
Customer c = new Customer("Jack Spratt", "747-1674",
    "125 Bismark St, Pleasantville, NY 14850", 7008);
```

III
```
Customer c = new Customer("Jack Spratt", "747-1674",
    new Address("125 Bismark St", "Pleasantville", "NY", 14850),
    7008);
```

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

11. Consider an AllCustomers class that has private instance variable

```
private Customer[] custList;
```

Given the ID number of a particular customer, a method of the class, locate, must find the correct Customer record and return the name of that customer. Here is the method locate:

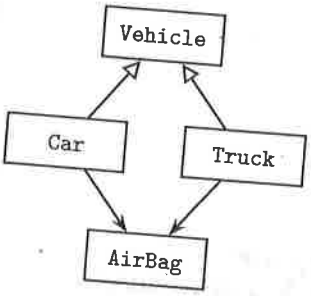
```
/** Precondition: custList contains a complete list of Customer objects.
 *  @param idNum the ID number for a Customer
 *  @return the name of the customer with the specified idNum
 */
public String locate(int idNum)
{
    for (Customer c : custList)
        if (c.getID() == idNum)
            return c.getName();
    return null;      //idNum not found
}
```

A more efficient algorithm for finding the matching Customer object could be used if

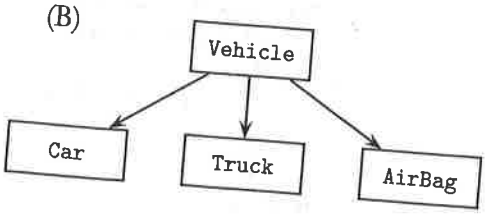
(A) Customer objects were in alphabetical order by name.
(B) Customer objects were sorted by phone number.
(C) Customer objects were sorted by ID number.
(D) the custList array had fewer elements.
(E) the Customer class did not have an Address data member.

20. Consider a program that deals with various components of different vehicles. Which of the following is a reasonable representation of the relationships among some classes that may comprise the program? Note that an open up-arrow denotes an inheritance relationship and a down-arrow denotes a composition relationship.

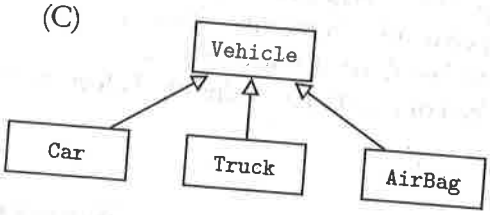
(A)



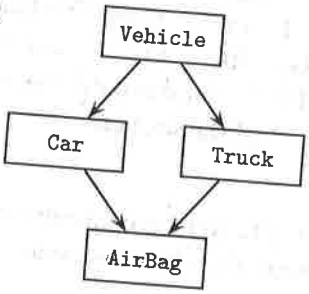
(B)



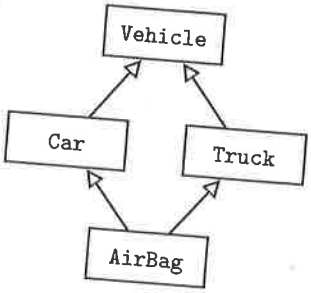
(C)



(D)



(E)



Questions 25–26 are based on the following class declaration:

```
public class AutoPart
{
    private String description;
    private int partNum;
    private double price;

    public AutoPart(String desc, int pNum, double aPrice)
    { /* implementation not shown */ }

    public String getDescription()
    { return description; }

    public int getPartNum()
    { return partNum; }

    public double getPrice()
    { return price; }

    //Other methods are not shown.
    //There is no compareTo method.
}
```

25.

25. This question refers to the `findCheapest` method below, which occurs in a class that has an array of `AutoPart` as one of its private data fields:

```
private AutoPart[] allParts;
```

The `findCheapest` method examines an array of `AutoPart` and returns the part number of the `AutoPart` with the lowest price whose description matches the `partDescription` parameter. For example, several of the `AutoPart` elements may have `"headlight"` as their description field. Different headlights will differ in both price and part number. If the `partDescription` parameter is `"headlight"`, then `findCheapest` will return the part number of the cheapest headlight.

```
/** Precondition: allParts contains at least one element whose
 *                description matches partDescription.
 *  @param partDescription the description of a part in allParts
 *  @return the part number of the cheapest AutoPart
 *  whose description matches partDescription
 */
public int findCheapest(String partDescription)
{
    AutoPart part = null;        //AutoPart with lowest price so far
    double min = LARGE_VALUE;    //larger than any valid price
    for (AutoPart p : allParts)
    {
        /* more code */
    }
    return part.getPartNum();
}
```

Which of the following replacements for `/* more code */` will find the correct part number?

I
```
if (p.getPrice() < min)
{
    min = p.getPrice();
    part = p;
}
```

II
```
if (p.getDescription().equals(partDescription))
    if (p.getPrice() < min)
    {
        min = p.getPrice();
        part = p;
    }
```

III
```
if (p.getDescription().equals(partDescription))
    if (p.getPrice() < min)
        return p.getPartNum();
```

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

26. Consider the following method:

```
/** Precondition: st1 and st2 are distinct String objects.
 *  @return smaller of st1 and st2
 */
public static String min(String st1, String st2)
{
   .
   .
   .
   if (st1.compareTo(st2) < 0)
      return st1;
   else
      return st2;
}
```

A method in the same class has these declarations:

```
AutoPart p1 = new AutoPart(<suitable values>);
AutoPart p2 = new AutoPart(<suitable values>);
```

Which of the following statements will cause an error?

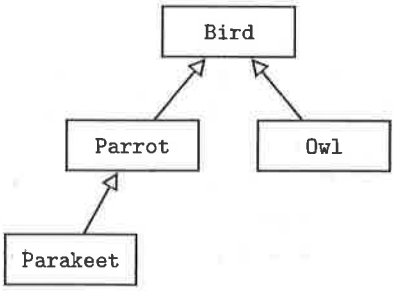
I
```
System.out.println(min(p1.getDescription(),
p2.getDescription()));
```

II
```
System.out.println(min(((String) p1).getDescription(),
    ((String) p2).getDescription()));
```

III
```
System.out.println(min(p1, p2));
```

(A) None
(B) I only
(C) II only
(D) III only
(E) II and III only

32. Consider the following hierarchy of classes:



Assuming that each class has a valid default constructor, which of the following declarations in a client program are correct?

I
```
Bird b1 = new Parrot();
Bird b2 = new Parakeet();
Bird b3 = new Owl();
```

II
```
Parakeet p = new Parrot();
Owl o = new Bird();
```

III
```
Parakeet p = new Bird();
```

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

16. Consider the following two classes.

```
public class Performer
{
    public void act()
    {
        System.out.print(" bow");
        perform();
    }

    public void perform()
    {
        System.out.print(" act");
    }
}

public class Singer extends Performer
{
    public void act()
    {
        System.out.print(" rise");
        super.act();
        System.out.print(" encore");
    }

    public void perform()
    {
        System.out.print(" aria");
    }
}
```

Suppose the following declaration appears in a class other than `Performer` or `Singer`:

```
Performer p = new Singer();
```

What is printed as a result of the call `p.act();`?

(A) `rise bow aria encore`
(B) `rise bow act encore`
(C) `rise bow act`
(D) `bow act aria`
(E) `bow aria encore`

Use the program description below for Questions 17–19.

A car dealer needs a program that will maintain an inventory of cars on his lot. There are three types of cars: sedans, station wagons, and SUVs. The model, year, color, and price need to be recorded for each car, plus any additional features for the different types of cars. The program must allow the dealer to

- Add a new car to the lot.
- Remove a car from the lot.
- Correct any data that's been entered.
- Display information for any car.

17. The programmer decides to have these classes: `Car`, `Inventory`, `Sedan`, `SUV`, and `StationWagon`. Which statement is *true* about the relationships between these classes and their attributes?

I There are no inheritance relationships between these classes.
II The `Inventory` class *has-a* list of `Car` objects.
III The `Sedan`, `StationWagon`, and `SUV` classes are independent of each other.

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

18. Suppose that the programmer decides to have a `Car` class and an `Inventory` class. The `Inventory` class will maintain a list of all the cars on the lot. Here are some of the methods in the program:

```
addCar          //adds a car to the lot
removeCar       //removes a car from the lot
displayCar      //displays all the features of a given car
setColor        //sets the color of a car to a given color
                //May be used to correct data
getPrice        //returns the price of a car
displayAllCars  //displays features for every car on the lot
```

In each of the following, a class and a method are given. Which is the *least* suitable choice of class to be responsible for the given method?

(A) `Car, setColor`
(B) `Car, removeCar`
(C) `Car, getPrice`
(D) `Car, displayCar`
(E) `Inventory, displayAllCars`

19. Suppose `Car` is a superclass and `Sedan`, `StationWagon`, and `SUV` are subclasses of `Car`. Which of the following is the most likely method of the `Car` class to be overridden by at least one of the subclasses (`Sedan`, `StationWagon`, or `SUV`)?

(A) `setColor(newColor)` `//sets color of Car to newColor`
(B) `getModel()` `//returns model of Car`
(C) `displayCar()` `//displays all features of Car`
(D) `setPrice(newPrice)` `//sets price of Car to newPrice`
(E) `getYear()` `//returns year of Car`

Questions 23–25 are based on the three classes below:

```
public class Employee
{
    private String name;
    private int employeeNum;
    private double salary, taxWithheld;

    public Employee(String aName, int empNum, double aSalary,
        double aTax)
    { /* implementation not shown */ }

    /** @return pre-tax salary */
    public double getSalary()
    { return salary; }

    public String getName()
    { return name; }

    public int getEmployeeNum()
    { return employeeNum; }

    public double getTax()
    { return taxWithheld; }

    public double computePay()
    { return salary - taxWithheld; }
}

public class PartTimeEmployee extends Employee
{
    private double payFraction;

    public PartTimeEmployee(String aName, int empNum, double aSalary,
        double aTax, double aPayFraction)
    { /* implementation not shown */ }

    public double getPayFraction()
    { return payFraction; }

    public double computePay()
    { return getSalary() * payFraction - getTax();}
}

public class Consultant extends Employee
{
    private static final double BONUS = 5000;

    public Consultant(String aName, int empNum, double aSalary,
        double aTax)
    { /* implementation not shown */ }

    public double computePay()
    { /* implementation code */ }
}
```

23. The `computePay` method in the `Consultant` class redefines the `computePay` method of the `Employee` class to add a bonus to the salary after subtracting the tax withheld. Which represents correct `/* implementation code */` of `computePay` for `Consultant`?

I
```
return super.computePay() + BONUS;
```

II
```
super.computePay();
return getSalary() + BONUS;
```

III
```
return getSalary() - getTax() + BONUS;
```

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

24. Consider these valid declarations in a client program:

```
Employee e = new Employee("Noreen Rizvi", 304, 65000, 10000);
Employee p = new PartTimeEmployee("Rafael Frongillo", 287, 40000,
    7000, 0.8);
Employee c = new Consultant("Dan Lepage", 694, 55000, 8500);
```

Which of the following method calls will cause an error?

(A) `double x = e.computePay();`
(B) `double y = p.computePay();`
(C) `String n = c.getName();`
(D) `int num = p.getEmployeeNum();`
(E) `double g = p.getPayFraction();`

25. Consider the `writePayInfo` method:

```
/** Writes Employee name and pay on one line. */
public static void writePayInfo(Employee e)
{ System.out.println(e.getName() + " " + e.computePay()); }
```

The following piece of code invokes this method:

```
Employee[] empList = new Employee[3];
empList[0] = new Employee("Lila Fontes", 1, 10000, 850);
empList[1] = new Consultant("Momo Liu", 2, 50000, 8000);
empList[2] = new PartTimeEmployee("Moses Wilks", 3, 25000, 3750,
    0.6);
for (Employee e : empList)
    writePayInfo(e);
```

What will happen when this code is executed?

(A) A list of employees' names and corresponding pay will be written to the screen.

(B) A `NullPointerException` will be thrown.

(C) A `ClassCastException` will be thrown.

(D) A compile-time error will occur, with the message that the `getName` method is not in the `Consultant` class.

(E) A compile-time error will occur, with the message that an instance of an `Employee` object cannot be created.

29. Assume that a `Book` class has a `compareTo` method where, if `b1` and `b2` are `Book` objects, `b1.compareTo(b2)` is a negative integer if `b1` is less than `b2`, a positive integer if `b1` is greater than `b2`, and `0` if `b1` equals `b2`. The following method is intended to return the index of the "smallest" book, namely the book that would appear first in a sorted list of `Book` objects.

```
/** Precondition:
 *   - books is initialized with Book objects.
 *   - books.length > 0.
 */
public static int findMin(Book[] books)
{
    int minPos = 0;
    for (int index = 1; index < books.length; index++)
    {
        if ( /* condition */ )
        {
            minPos = index;
        }
    }
    return minPos;
}
```

Which of the following should be used to replace `/* condition */` so that `findMin` works as intended?

(A) `books[minPos] > books[index]`
(B) `books[index] > books[minPos]`
(C) `books[index].compareTo(books[minPos]) > 0`
(D) `books[index].compareTo(books[minPos]) >= 0`
(E) `books[index].compareTo(books[minPos]) < 0`

30. Refer to the static method `removeNegs` shown below.

```
/** Precondition: list is an ArrayList<Integer>.
 *  Postcondition: All negative values have been removed from list.
 *  @param list the list of Integer objects
 */
public static void removeNegs(List<Integer> list)
{
    int index = 0;
    while (index < list.size())
    {
        if (list.get(index).intValue() < 0)
        {
            list.remove(index);
        }
        index++;
    }
}
```

For which of the following lists will the method *not* work as intended?

(A) `6 -1 -2 5`
(B) `-1 2 -3 4`
(C) `2 4 6 8`
(D) `-3`
(E) `1 2 3 -8`

14) Consider the following code segment.

```
public class MyClass
{
  public MyClass()
  {
    // code
  }

  // more code
}
```

To instantiate `MyClass`, which of the following statements should you use?

(A) `MyClass mc = new MyClass();`

(B) `MyClass mc = MyClass();`

(C) `MyClass mc = new MyClass;`

(D) `MyClass mc = MyClass;`

(E) None of the above

15) Consider the following code segment.

```
submarine.drive(depth);
```

Which of the following statements must be true?

(A) `submarine` is a class and `drive` is a method

(B) `drive` is a class and `submarine` is a method

(C) `submarine` is a variable

(D) `drive` is a variable

(E) None of the above

For Questions 16 - 18, refer to the following code.

```
public class Person
{
  private String name;
  private int age;

  public person(String name, int age)
  {
    this.name = name;
    this.age = age;
  }
  public int getName()
  {
    return name;
  }
  public int getAge()
  {
    return age;
  }
  public void setName(String name)
  {
    this.name = Name;
  }
}
```

16) Which of the following code segments correctly accesses the name attribute?

I.
```
Person p = new Person("zeshan", 14);
String name = p.getName();
```

II.
```
Person p = new Person("zeshan", 14);
String name;
name = p.getName();
```

III.
```
Person p = new Person(zeshan, 14);
String name = p.getName();
```

(A) I only

(B) II only

(C) I and II

(D) I, II and III

(E) III only

17) Which of the following code segments correctly sets "zeshan" to be the name of the AccessPerson object?

I.
```
AccessPerson.setName(zeshan);
```

II.
```
AccessPerson.setName("zeshan");
```

III.
```
Person AccessPerson = new Person(ali, 14);
AccessPerson.setName("zeshan");
```

(A) I only

(B) I and III only

(C) I, II and III

(D) III only

(E) II only

18) Which of the following code segments does not compile?

I.
```
Person p = new Person(zeshan, 14);
String name = p.getName();
```

II.
```
Person p = new Person(zeshan, 14);
name = p.getName();
```

III.
```
Person p = new Person("zeshan", 14);
String name = p.getName();
```

(A) I, II and III

(B) I only

(C) II only

(D) III only

(E) I and II

For Questions 21 - 22, refer to the following code segment.

```
Object o = "abc";
boolean b = o.equals("a,b,c");

Object o2 = b;
String t = (String) o;

System.out.println(o2);
System.out.println(o == t);
```

21) What will be the result of the execution of the following statement?

```
System.out.println(o2);
```

I. `true`

II. `false`

III. Compile time error

(A) II only

(B) I only

(C) III only

(D) I and II

(E) None

22) What will be the result of the execution of the following statement?

```
System.out.println(o == t);
```

I. `true`

II. `false`

III. Compile time error

(A) I only

(B) II only

(C) III only

(D) I and III

(E) None

23) Consider the following code segment.

```
1  Integer i = new Integer(0);
2  if (!(i.equals(0)))
3  {
4    System.out.println(i + " does not equal to " + 0);
5  }
6  else
7  {
8    System.out.println("Done");
9  }
```

Which of the following is true?

I. The code will not compile

II. Done will be printed out

III. There is an error on Line 1

(A) I only

(B) I and III only

(C) II only

(D) I and II only

(E) III only

25) Consider the following class declarations.

```
public class ClassOne
{
  private int x;

  public ClassOne()
  {
    x = 0;
  }

  public ClassOne(int y)
  {
    x = y;
  }
}


public class ClassTwo
{
  public ClassTwo()
  {
    super(0);
  }
}
```

Which of the following statements will not successfully compile?

(A) `ClassOne c1 = new ClassOne();`

(B) `ClassOne c2 = new ClassOne(5);`

(C) `ClassOne c3 = new ClassTwo();`

(D) `ClassTwo c4 = new ClassTwo();`

(E) `ClassTWo c5 = new ClassTwo(5);`

ıd 5.5?

11) Consider the following code segment with parent class A and child class B.

```
public class A
{
  int i;
}


public class B extends A
{
  int j;
  public void display()
  {
    super.i = j + 1;
    System.out.println(j + " " + i);
  }
}


public class Inheritance
{
  public static void main(String args[])
  {
    B obj = new B();
    obj.i = 1;
    obj.j = 2;
    obj.display();
  }
}
```

What will the output be if the main method of Inheritance is executed?

(A) 2 2

(B) 3 3

(C) 2 3

(D) 3 2

(E) 4 2