12. Which statement about abstract classes and interfaces is *false*?
    (A) An interface cannot implement any non-default instance methods, whereas an abstract class can.
    (B) A class can implement many interfaces but can have only one superclass.
    (C) An unlimited number of unrelated classes can implement the same interface.
    (D) It is not possible to construct either an abstract class object or an interface object.
    (E) All of the methods in both an abstract class and an interface are public.

19. Consider the `Computable` interface below for performing simple calculator operations:

```
public interface Computable
{
    /** Return this Object + y. */
    Object add(Object y);

    /** Return this Object - y. */
    Object subtract(Object y);

    /** Return this Object * y. */
    Object multiply(Object y);
}
```

Which of the following is the *least* suitable class for implementing `Computable`?

(A) `LargeInteger` `//integers with 100 digits or more`

(B) `Fraction` `//implemented with numerator and`
`//denominator of type int`

(C) `IrrationalNumber` `//nonrepeating, nonterminating decimal`

(D) `Length` `//implemented with different units, such`
`//as inches, centimeters, etc.`

(E) `BankAccount` `//implemented with balance`

Refer to the `Player` interface shown below for Questions 20–23.

```
public interface Player
{
    /** Return an integer that represents a move in a game. */
    int getMove();

    /** Display the status of the game for this Player after
     *  implementing the next move. */
    void updateDisplay();
}
```

20. `HumanPlayer` is a class that implements the `Player` interface. Another class, `SmartPlayer`, is a subclass of `HumanPlayer`. Which statement is *false*?
    (A) `SmartPlayer` automatically implements the `Player` interface.
    (B) `HumanPlayer` must contain implementations of both the `updateDisplay` and `getMove` methods, or be declared as abstract.
    (C) It is not possible to declare a reference of type `Player`.
    (D) The `SmartPlayer` class can override the methods `updateDisplay` and `getMove` of the `HumanPlayer` class.
    (E) A method in a client program can have `Player` as a parameter type.

21. A programmer plans to write programs that simulate various games. In each case he will have several classes, each representing a different kind of competitor in the game, such as `ExpertPlayer`, `ComputerPlayer`, `RecklessPlayer`, `CheatingPlayer`, `Beginner`, `IntermediatePlayer`, and so on. It may or may not be suitable for these classes to implement the `Player` interface, depending on the particular game being simulated. In the games described below, which is the *least* suitable for having the competitor classes implement the given `Player` interface?

    (A) High-Low Guessing Game: The computer thinks of a number and the competitor who guesses it with the least number of guesses wins. After each guess, the computer tells whether its number is higher or lower than the guess.

    (B) Chips: Start with a pile of chips. Each player in turn removes some number of chips. The winner is the one who removes the final chip. The first player may remove any number of chips, but not all of them. Each subsequent player must remove at least one chip and at most twice the number removed by the preceding player.

    (C) Chess: Played on a square board of 64 squares of alternating colors. There are just two players, called White and Black, the colors of their respective pieces. The players each have a set of pieces on the board that can move according to a set of rules. The players alternate moves, where a move consists of moving any one piece to another square. If that square is occupied by an opponent's piece, the piece is captured and removed from the board.

    (D) Tic-Tac-Toe: Two players alternate placing "X" or "O" on a 3×3 grid. The first player to get three in a row, where a row can be vertical, horizontal, or diagonal, wins.

    (E) Battleships: There are two players, each with a 10×10 grid hidden from his opponent. Various "ships" are placed on the grid. A move consists of calling out a grid location, trying to "hit" an opponent's ship. Players alternate moves. The first player to sink his opponent's fleet wins.

Consider these declarations for Questions 22 and 23:

```
public class HumanPlayer implements Player
{
    private String name;

    //Constructors not shown ...

    //Code to implement getMove and updateDisplay not shown ...

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

public class ExpertPlayer extends HumanPlayer
{
    private int rating;

    //Constructors not shown ...

    public int compareTo(ExpertPlayer expert)
    { /* implementation not shown */ }
}
```

22. Which code segment in a client program will cause an error?

I
```
Player p1 = new HumanPlayer();
Player p2 = new ExpertPlayer();
int x1 = p1.getMove();
int x2 = p2.getMove();
```

II
```
int x;
Player c1 = new ExpertPlayer(/* correct parameter list */);
Player c2 = new ExpertPlayer(/* correct parameter list */);
if (c1.compareTo(c2) < 0)
    x = c1.getMove();
else
    x = c2.getMove();
```

III
```
int x;
HumanPlayer h1 = new HumanPlayer(/* correct parameter list */);
HumanPlayer h2 = new HumanPlayer(/* correct parameter list */);
if (h1.compareTo(h2) < 0)
    x = h1.getMove();
else
    x = h2.getMove();
```

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

23. Which of the following is correct implementation code for the `compareTo` method in the `ExpertPlayer` class?

I
```
if (rating == expert.rating)
    return 0;
else if (rating < expert.rating)
    return -1;
else
    return 1;
```

II
```
return rating - expert.rating;
```

III
```
if (getName().equals(expert.getName()))
    return 0;
else if (getName().compareTo(expert.getName()) < 0)
    return -1;
else
    return 1;
```

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

24. Which of the following classes is the least suitable candidate for containing a `compareTo` method?

(A)
```
public class Point
{
    private double x;
    private double y;

    //various methods follow
        ...
}
```

(B)
```
public class Name
{
    private String firstName;
    private String lastName;

    //various methods follow
        ...
}
```

(C)
```
public class Car
{
    private int modelNumber;
    private int year;
    private double price;

    //various methods follow
        ...
}
```

(D)
```
public class Student
{
    private String name;
    private double gpa;

    //various methods follow
        ...
}
```

(E)
```
public class Employee
{
    private String name;
    private int hireDate;
    private double salary;

    //various methods follow
        ...
}
```

25. A programmer has the task of maintaining a database of students of a large university. There are two types of students, undergraduates and graduate students. About a third of the graduate students are doctoral candidates.

All of the students have the same personal information stored, like name, address, and phone number, and also student information like courses taken and grades. Each student's GPA is computed, but differently for undergraduates and graduates. The doctoral candidates have information about their dissertations and faculty advisors.

The programmer will write a Java program to handle all the student information. Which of the following is the best design, in terms of programmer efficiency and code reusability? Note: `{ ... }` denotes class code.

(A)
```
public interface Student { ...}
public class Undergraduate implements Student { ... }
public class Graduate implements Student { ... }
public class DocStudent extends Graduate { ... }
```

(B)
```
public abstract class Student { ...}
public class Undergraduate extends Student { ... }
public class Graduate extends Student { ... }
public class DocStudent extends Graduate { ... }
```

(C)
```
public class Student { ...}
public class Undergraduate extends Student { ... }
public class Graduate extends Student { ... }
public class DocStudent extends Graduate { ... }
```

(D)
```
public abstract class Student { ...}
public class Undergraduate extends Student { ... }
public class Graduate extends Student { ... }
public class DocStudent extends Student { ... }
```

(E)
```
public interface PersonalInformation { ... }
public class Student implements PersonalInformation { ...}
public class Undergraduate extends Student { ... }
public abstract class Graduate extends Student { ... }
public class DocStudent extends Graduate { ... }
```

26. Consider the `Orderable` interface and the partial implementation of the `Temperature` class defined below:

```
public interface Orderable
{
    /** Returns -1, 0, or 1 depending on whether the implicit
     *  object is less than, equal to, or greater than other.
     */
    int compareTo (Object other);
}

public class Temperature implements Orderable
{
    private String scale;
    private double degrees;

    //default constructor
    public Temperature ()
    { /* implementation not shown */ }

    //constructor
    public Temperature(String tempScale, double tempDegrees)
    { /* implementation not shown */ }

    public int compareTo(Object obj)
    { /* implementation not shown */ }

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

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

Here is a program that finds the lowest of three temperatures:

```
public class TemperatureMain
{
    /** Find smaller of objects a and b. */
    public static Orderable min(Orderable a, Orderable b)
    {
        if (a.compareTo(b) < 0)
             return a;
        else
             return b;
    }

    /** Find smallest of objects a, b, and c. */
    public static Orderable minThree(Orderable a,
            Orderable b, Orderable c)
    {
        return min(min(a, b), c);
    }

    public static void main(String[] args)
    {
        /* code to test minThree method */
    }
}
```

Which are correct replacements for /* *code to test* `minThree` *method* */?

I
```
Temperature t1 = new Temperature("C", 85);
Temperature t2 = new Temperature("F", 45);
Temperature t3 = new Temperature("F", 120);
System.out.println("The lowest temperature is " +
        minThree(t1, t2, t3));
```

II
```
Orderable c1 = new Temperature("C", 85);
Orderable c2 = new Temperature("F", 45);
Orderable c3 = new Temperature("F", 120);
System.out.println("The lowest temperature is " +
        minThree(c1, c2, c3));
```

III
```
Orderable c1 = new Orderable("C", 85);
Orderable c2 = new Orderable("F", 45);
Orderable c3 = new Orderable("F", 120);
System.out.println("The lowest temperature is " +
        minThree(c1, c2, c3));
```

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

27. A certain interface provided by a Java package contains just a single method:

```
public interface SomeName
{
    int method1(Object o);
}
```

A programmer adds some functionality to this interface by adding another abstract method to it, `method2`:

```
public interface SomeName
{
    int method1(Object ob1);
    void method2(Object ob2);
}
```

As a result of this addition, which of the following is true?

(A) A `ClassCastException` will occur if `ob1` and `ob2` are not compatible.

(B) All classes that implement the original `SomeName` interface will need to be rewritten because they no longer implement `SomeName`.

(C) A class that implements the original `SomeName` interface will need to modify its declaration as follows:

```
public class ClassName implements SomeName extends method2
{    ...
```

(D) `SomeName` will need to be changed to an abstract class and provide implementation code for `method2`, so that the original and upgraded versions of `SomeName` are compatible.

(E) Any new class that implements the upgraded version of `SomeName` will not compile.