C1 (A1.5). Consider the following description.
A forest has trees. A white pine is a pine which is a tree.

Which of the following partial declarations would be the best choice for representing the relationships among these things?

A.
```
public class Forest
   public class Tree extends Forest
   public class Pine extends Tree
   public class WhitePine extends Pine
```

B.
```
public class WhitePine
   public class Pine extends WhitePine
   public class Tree extends Pine
   public class Forest extends Tree
```

C.
```
public class WhitePine
   public class Pine extends WhitePine
   public class Tree extends Pine

   public class Forest
   {
      private ArrayLit<Pine> myPines;
      private ArrayList<WhitePine> myWhitePines;
       . . .
   }
```

D.
```
public class Tree
   public class Pine extends Tree
   public class WhitePine extends Pine

   public class Forest
   {
      private List<Tree> myTrees;
       . . .
   }
```

E.
```
public class Forest

   public class Tree
   {
      private Forest myForest;
       . . .
   }
   public class Pine extends Tree
   public class WhitePine extends Pine
```

C2 (A2.7). Consider the following description
An estate has a house and has a garden. A garden has flowers. A crocus is a flower and a daffodil is a flower.

Which of the following partial declarations would be the best choice for representing the relationships among these things?

A.
```
public class House
public class Garden
{
  private ArrayList<Flower> myFlowers;
    . . .
}

public class Estate
{
   private House myHouse;
   private Garden myGarden;
    . . .
}
public class Flower
public class Crocus extends Flower
public class Daffodil extends Flower
```

B.
```
public class House
public class Garden
{
  private ArrayList<Flower> myFlowers;
    . . .
}
public class Estate extends House
{
   private Garden myGarden;
    . . .
}
public class Flower
public class Crocus extends Flower
public class Daffodil extends Flower
```

C.
```
public class House
public class Garden
{
  private ArrayList<Flower> myFlowers;
    . . .
}

public class Estate
{
   private House myHouse;
   private Garden myGarden;
    . . .
}

public class Flower
{
   private boolean isCrocus;
   private boolean isDaffodil;
}
```

D.
```
public class House
public class Garden
{
  private ArrayList<Flower> myFlowers;
    . . .
```

```
   }

   public class Estate extends House
   {
      private Garden myGarden;
       . . .

   }

   public class Flower
   {
      private boolean isCrocus;
      private boolean isDaffodil;
   }
```

E.
```
   public class House
   public class Garden

   public class Estate
   {
      private House myHouse;
      private Garden myGarden;
       . . .
   }
   public class Flower
   {
      private Garden myGarden;
   }

   public class Crocus extends Flower
   public class Daffodil extends Flower
```

C3 (A1.35). Consider the following partial declarations.

```
public class Location
{
  public Location(int xCoord, int yCoord)
  . . .
}
public class Color
{
  public Color(int redVal, int greenVal, int blueVal)
  . . .
}
public class Widget
{
  private Location myLoc;

  public Widget(int x, int y)
  {
```

```
      myLoc = new Location(x, y);
  }
  . . .
}
public class Thingy extends Widget
{
  private Color myColor;

  public Thingy(int x, int y, Color col)
  {
    super(x, y);
    myColor = col;
  }
  . . .
}
```

Assume that the following statement appears in a client program.

```
Widget widg = new Thingy(100, 100, new Color(100, 100, 100));
```

Which of the following best describes the order on which the constructors will complete execution?

A. Color, Location, Widget, Thingy
B. Widget, Thingy, Location, Color
C. Color, Widget, Location, Thingy
D. Thingy, Color, Widget, Location
E. Color, Thingy, Location, Widget

C4 (none). Consider the following partial declarations.

```
public class Location
{
  public Location(int xCoord, int yCoord)
  . . .
}
public class Color
{
  public Color(int redVal, int greenVal, int blueVal)
  . . .
}
public class Widget
{
  private Location myLoc;

  public Widget(int x, int y)
  {
     myLoc = new Location(x, y);
  }
  . . .
}
public class Thingy extends Widget
{
  private Color myColor;

  public Thingy(int x, int y, Color col)
  {
    super(x, y);
    myColor = col;
  }
  . . .
}
```

Assume that the following statement appears in a client program.

```
Widget widg = new Thingy(100, 100, new Color(100, 100, 100));
```

Which of the following best describes the order on which the constructors will begin execution?

A. Color, Location, Widget, Thingy
B. Widget, Thingy, Location, Color
C. Color, Widget, Thingy, Location
D. Thingy, Color, Widget, Location
E. Color, Thingy, Widget, Location

C5 (A2.9). Assume that the following partial declarations have been made, with default constructors for the classes.

```
public interface Controller

public class Widget implements Controller

public class Thingy extends Widget
```

Consider the following declarations.

```
I.    Widget myThing = new Thingy();

II.   Controller myWidge = new Thingy();

III.  Controller myControl = new Controller();
```

Which of these declarations will compile correctly?

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

C6 (A1.6). Assume that the following partial declarations have been made, with default constructors for the classes.

```
public interface Controller;

public class Widget implements Controller;

public class Thingy extends Widget;
```

Consider the following declarations.

```
I.    Controller myControl = new Thingy();

II.   Controller myControl = new Controller();

III.  Thingy myWidge = new Widget();
```

Which of these declarations will compile correctly?

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

C7 (A2.4). Which of the following statements about an interface is true?

A. An interface can only have a default (no parameter) constructor.
B. An interface can only have instance variables that are private.
C. An interface can only have instance variables that are public.
D. An interface can only have methods that are private.
E. An interface can only have methods that are public.

C8 (A1.7). Which of the following statements about an abstract class is true?

A. An abstract class can only have a default (no parameter) constructor.
B. An abstract class cannot have instance variables.
C. An abstract class can only have methods that are public.
D. An abstract class can have some methods that are implemented and some that are not.
E. An abstract class can only have methods that are declared abstract and are not implemented.

C9 (A2.30). Consider the following declarations, where some lines have been numbered for reference.

```
      public interface Controller
      {
1       void doIt();
      }

      public class Widget implements Controller
      {
2       private double myValue;

        public Widget()
        {
          myValue = 10;
        }

        public void doIt()
        {
          System.out.println( 2.0 * myValue);
        }
      }

      public class Thingy extends Widget
      {
        public void doIt()
        {
3         System.out.println( 3.0 * myValue);
        }
      }

      public class Client
      {
        public void doSomething()
        {
4         Controller myControl = new Thingy();
          myControl.doIt();
        }
      }
```

You attempt to compile and run these classes (with a class that correctly calls the `Client doSomething` method). Which of the following best describes the outcome?

A. The code does not compile because in line 1 the `doIt` method is not declared public.
B. The code does not compile because the class `Widget` cannot have a private instance variables that is not already declared in the interface `Controller`, as it does in line 2.
C. The code does not compile because the variable `myValue` cannot be accessed within the class `Thingy`, as it is in line 3.
D. The code does not compile because an instance of class `Thingy` is not type-compatible with `Controller`, as indicated in line 4.
E. The code compiles and runs with the value 30 output when a call to `doSomething` is made.

C10 (A1.28). Consider the following declarations, where some lines have been numbered for reference.

```
      public interface Controller
      {
1       void doIt();
      }

2     public abstract class Widget implements Controller
      {
        private double myValue;

        public Widget()
        {
          myValue = 10;
        }

        public double value()
        {
          return myValue;
        }
      }

      public class Thingy extends Widget
      {
        public void doIt()
        {
3         System.out.println( 3.0 * value());
        }
      }

      public class Client
      {
        public void doSomething()
        {
4         Controller myControl = new Thingy();
          myControl.doIt();
        }
      }
```

You attempt to compile and run these classes (with a class that correctly calls the `Client doSomething` method). Which of the following best describes the outcome?

A. The code does not compile because in line 1 the `doIt` method is not declared public.
B. The code does not compile because in line 2 the class `Widget` implements `Controller`, but it has no implementation of the method `doIt`.
C. The code does not compile because the variable `myValue` cannot be accessed directly or indirectly within the class `Thingy`, as it is in line 3.
D. The code does not compile because an instance of class `Thingy` is not type-compatible with `Controller`, as indicated in line 4.
E. The code compiles and runs with the value 30.0 output when the call to `doSomething` is made.

C11 (A1.13). Consider the following declarations.

```
public class ColorBox
{
  public ColorBox()
  {
    System.out.print("black ");
  }

  public void showColor()
  {
    System.out.print("red ");
  }
}

public class BlueGreenBox extends ColorBox
{
  public BlueGreenBox()
  {
    System.out.print("blue ");
  }

  public void showColor()
  {
    super.showColor();
    System.out.print("green ");
  }
}
```

The following statements occur in a client method:

```
ColorBox box = new BlueGreenBox();
box.showColor();
```

What is printed when these two lines are executed?

A. black red
B. blue green
C. blue red green
D. blue black red green
E. black blue red green

C12 (A2.33). Consider the following declarations.

```
public class ColorBox
{
  public ColorBox()
  {
    System.out.print("black ");
  }

  public void showColor()
  {
    System.out.print("red ");
  }
}

public class BlueGreenBox extends ColorBox
{
  public BlueGreenBox()
  {
    System.out.print("blue ");
    super();
  }

  public void showColor()
  {
    System.out.print("green ");
    super.showColor();
  }
}
```

The following statements occur in a client method:

```
ColorBox box = new BlueGreenBox();
box.showColor();
```

What is happens when these classes and the client are compiled and executed?

A. `blue black green red` is printed
B. `black blue green red` is printed
C. `black blue black green red` is printed
D. `black blue black red green red` is printed
E. The classes and code fail to compile correctly

C13 (A2.13). Which of the following is the best reason to define an interface with several classes that implement the interface?

A. Each of the classes implementing the interface shares a set of instance variables defined within the interface.
B. Each of the classes implementing the interface can have part of its initialization involving common elements done by a constructor defined in the interface.
C. Each of the classes implementing the interface has all the methods specified in the interface and can therefore be passed as an actual parameter to a method calling for the interface type as parameter.
D. Each of the classes implementing the interface has only those methods specified by the interface and can therefore be stored in an array of elements of the interface type.
E. Each of the classes implementing the interface can be used interchangeably for assignment to any variable with type specified as any of the other classes implementing the interface.

C14 (A1.32). Which of the following is the best reason to define an abstract class with several classes that extend the abstract class?

A. All of the classes that extend the abstract class share some common responsibilities and attributes that are defined as methods and instance variables in the abstract class.
B. A class can only extend one concrete (non-abstract) class, but can also extend additional abstract classes that define another part of its responsibilities.
C. The abstract class defines all of the instance variables that can be used by the classes that extend it; each class that extends it can introduce new methods, but not new instance variable.
D. Each of the classes extending the abstract class has only those methods specified by the abstract class and can therefore be stored in an array of elements of the abstract class type.
E. Each of the classes implementing the abstract class can be used interchangeably for assignment to any variable with type specified as any of the other classes implementing the abstract class.

C15 (A3.5). Consider the following description. A Kennel has animals. Animals come and go from the kennel, so the number there varies. A dog is an animal. A cat is an animal. Which of the following partial declarations is the best choice for representing the relationships among these things?

A.
```
public class Dog
public class Cat
public class Kennel
{
  private ArrayList<Dog> myDogs;
  private ArrayList<Cat> myCats;
   ...
}
```

B.
```
public class Dog
public class Cat
public class Kennel
{
  private Animal[] myAnimals;
  private int count;
   ...
}
```

C.
```
public class Dog
public class Cat
public class Kennel
{
  private ArrayList<Animal> myAnimals;
   ...
}
```

D.
```
public abstract class Animal
public class Dog extends Animal
public class Cat extends Animal
public class Kennel
{
  private Animal[] myAnimals;
  private int count;
   ...
}
```

E.
```
public abstract class Animal
public class Dog extends Animal
public class Cat extends Animal
public class Kennel
{
  private ArrayList<Animal> myAnimals;
   ...
}
```

C16 (A3.35). Consider the following partial declarations.

```
public class Direction
{
  private int degrees;
  public Direction(int deg)
  {
    degrees = deg;
    System.out.println("direction ");
  }
  ...
}

public class Location
{
  private double xLoc, yLoc;
  public Location(double x, double y)
  {
    xLoc = x;
    yLoc = y;
    System.out.println("location ");
  }
  ...
}
```

```
public class Car
{
  private Direction dir;
  private Location loc;
  public Car(Direction d , Location l)
  {
    dir = d;
    loc = l;
    System.out.println("car");
  }
  ...
}

public class RaceCar extends Car
{
  private double speed;
  public RaceCar(Direction d, Location l,
                 double sp)
  {
    super(d, l);
    speed = sp;
    System.out.println("race-car");
  }
  ...
}
```

Consider the statement

```
RaceCar TSX = new RaceCar(new Direction(90), new Location(5, 8), 160);
```

What is printed when this statement is executed?

A. location direction race-car car
B. direction location race-car car
C. direction location car race-car
D. race-car car direction location
E. car race-car direction location

C17 (A3.6). Assume the following partial declarations have been made, with default (no parameter) constructors for the classes.

```
public abstract class Player
public class ComputerPlayer extends Player;
public class SmartComputerPlayer extends ComputerPlayer;
```

Consider the following declarations.

```
I.    ComputerPlayer playerA = new SmartComputerPlayer();
II.   Player playerB = new SmartComputerPlayer();
III.  Player playerC = new Player();
```

Which of these declarations will compile correctly?

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

C18 (A3.7) Which of the following statements is true about both an interface and an abstract class?

A. Both can only have a default (no parameter) constructor.
B. Both can not have any instance variables.
C. Both can have methods that are not implemented.
D. Both can have only public instance variables.
E. Both can extend another abstract class.

C19 (A3.28). Consider the following declarations.

```
public abstract class Player
{
  private int myCount;

  public Player()
  {  myCount = 0; }

  public int count()
  {  return myCount; }

  public void incCount(int amt)

  { myCount += amt; }

  public abstract int play();
}

public class Controller
{
  public void doIt()
  {
    Player p = new SmartPlayer();
    p.play();
  }
}
```

```
public class ComputerPlayer extends Player
{
  public ComputerPlayer()
  { super(); }
}


public class SmartPlayer extends ComputerPlayer
{
  public int play()
  {
    incCount(2);
    return count();
  }
}
```

You compile and run these classes with a driver class that correctly calls the `Controller` `doIt` method. Which of the following best describes the outcome?

A. The code does not compile because an abstract class cannot have an instance variable.
B. The code does not compile because an abstract class cannot implement a method.
C. The code does not compile because a subclass cannot access an instance variable in its superclass, either directly or indirectly.
D. The code does not compile because the class ComputerPlayer does not implement the method play and is not declared abstract.
E. The code does not compile because the class SmartPlayer does not define a default (no parameter) constructor.

C20 (A3.13). Consider the following declarations.

```
public class NoiseMaker
{
  public NoiseMaker(String sound)
  {  System.out.print(sound); }

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

public class LoudNoiseMaker extends NoiseMaker
{
  public LoudNoisemaker(String sound)
  { super(sound); }

  public void makeNoise()
  {
    super.makeNoise();
    super.makeNoise();
  }
}
```

The following method is in a client class.

```
public void doIt()
{
  NoiseMaker boomBox = new LoudNoiseMaker("ring");
  boomBox.makeNoise();
}
```

What happens when these classes are compiled and the method `doIt` is executed?

A. "ring boom" is printed.
B. "ring boom boom" is printed.
C. "boom ring" is printed.
D. "boom ring boom" is printed.
E. The code fails to compile because there are <u>two</u> calls to `super.makeNoise()` in the second `makeNoise` method.

C21. (A1.15)
Consider the following declarations.

```
I.    List<int> = new ArrayList<int>;
II.   List<Integer> = new ArrayList<Integer>;
III.  List<Integer> = new List<Integer>;
```

Which of these declarations is correct?

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

C22 (A3.17)
Consider the following declarations.

I. `ArrayList<Integer> = new ArrayList<Integer>();`
II. `List<Integer> = new ArrayList<Integer>();`
III. `List<Integer> = new List<Integer>();`

Which of these will compile correctly?

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