AP CS Unit 7: Interfaces
Exercises

Assume all code compiles unless otherwise suggested.

```
public interface Face{
    public void m1(int n);
    public int m2();
}
```

1. The Nose class ...
(a) compiles.
b) will not compile because the m1 method parameter should be named *n*, not *x*.

```
public class Nose implements Face{
    public void m1(int x){
        System.out.println( x );
    }
    public int m2(){
        return 7;
    }
}
```

2. The Ears class ...
a) compiles.
(b) will not compile because it needs to implement all the methods of the Face interface.

```
public class Ears implements Face{
    public void m1(int n){
        n = n + 4;
    }
}
```

3. The Eyes class ...
(a) compiles.
b) will not compile because it has a method that is not defined by the interface.

```
public class Eyes implements Face{
    public void m1(int x){
        System.out.println( x );
    }
    public int m2(){
        return 7;
    }
    public String hey(){
        return "hey";
    }
}
```

4. The Mouth class ...
a) compiles.
b) will not compile because method m1 is being used as a mutator method.
(c) will not compile because method m2 is missing a body.

Yes or must create class as abstract!

```
public class Mouth implements Face{
    private int z = 11;

    public void m1(int n){
        z = n;
    }
    public int m2();
}
```

5. Methods in an interface cannot be private.	(True)	False
6. If a class implements an interface, then it must define at least one, but not necessarily all, of the methods listed by the interface.	True	(False)
7. An interface may include implementing code.	True	(False)
8. The data type of parameter cannot be an interface.	True	(False)

9. Select the TRUE statement(s).

a) This code will not compile because a method cannot specify an interface as a parameter.

b) This code will compile if the Beans class implements the Commodity interface.

c) The keyword static is not required for methodA.

d) This code will compile as long as the Beans class has a getPrice method. The Beans class does not have to define a setPrice method.

```
public class Testing {
    public static void main( String [] args ) {
        Beans b = new Beans( 12 );
        methodA( b );
    }

    public static void methodA( Commodity c ) {
        System.out.println( c.getPrice() );
    }
}

// separate file
public interface Commodity {
        public double getPrice();
        public void setPrice( double c );
}
```

10. Select the TRUE statement(s). If this line compiles, then

a) ABC may refer to an interface and XYZ a class that implements that interface.

b) XYZ may refer to an interface and ABC a class that implements that interface.

c) if ABC is an interface, methodY must be a method defined by that interface. or it would have to be cast

d) if ABC is a class, methodY must be a method defined by that class.

e) whatever ABC is, XYZ must be a class that has, or inherits, a methodY.

```
ABC x = new XYZ();
x.methodY();
```

11. What is displayed?

lime
grape
peach

```
List<String> x = new ArrayList<String>();
x.add( "grape" );
x.add( 0, "lime" );
x.add( "peach" );

for ( int n = 0; n < x.size(); n++ ) {
        String s = x.get(n);
        System.out.println( s );
}
```

lime grape peach

12. What is displayed?

111
77
65
404

```
public void met( List<Integer> x ){
        x.add( 404 );
        x.add( 0, 65 );
        x.add( 0, 111 );
        x.set( 1, 77 );

        for ( Integer i : x )
                System.out.println( i );
}
```

Autoboxing and unboxing take place here.

111 65 404
↑
77

13. Select the TRUE statement.
a) This compiles because List objects are ArrayList objects.
b) This does not compile because not all List objects are ArrayList objects.
	public ArrayList<String> m1(){
	List<String> x = new ArrayList<String>();
	return x;
	}
	List 'is-a' Arraylist? NO

14. Select the TRUE statement.
a) This compiles because ArrayList objects are List objects.
b) This does not compile because not all ArrayList objects are List objects.
	public List<String> m2(){
	ArrayList<String> x = new ArrayList<String>();
	return x;
	}

15. What is displayed?

a → G → E → T
<change>
a → E → T → G

print E T G

a → T G E

print T G E

```
import java.util.*;

public class Runner {
  public static void main(String[] args) {
      ArrayList<String> a = new ArrayList<String>();
      a.add( "G" );
      a.add( "E" );
      a.add( "T" );

      change( a );
      for ( String s : a )
          System.out.print( s + "  ");
      System.out.println();

      change( a );
      for ( String s : a )
          System.out.print( s + "  ");
  }

  public static void change( List<String> x ){
      String str = x.remove(0);
      x.add( str );
  }
}
```

16. What is displayed?

x → 0 X 1 0

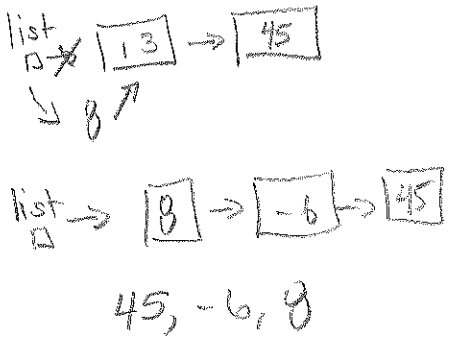
0 1 0

```
List <String> x = new ArrayList<String>();
x.add( "0" );
x.add( "1" );
x.add( "1" );
x.add( "0" );

for ( int n = 0; n < x.size(); n++ ) {
        String s = x.get(n);
        if ( s.equals( "1" ) )
                x.remove( n );
}

for ( int n = 0; n < x.size(); n++ )
        System.out.println( x.get( n ) );
```

important!

n: 0 1 2
s: 0 1 0

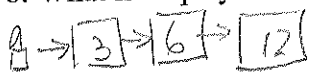
17. What is displayed?



45, -6, 8

```
List<Integer> list = new ArrayList<Integer>();
list.add( 13 );
list.add( 45 );
list.add( 0, 8 );

System.out.println( list.set( 1, -6 ) );      13

for ( int n = list.size() - 1; n >= 0; n-- )
        System.out.print( list.remove(n) + ", " );

System.out.println( "\n----------" );
System.out.println( list.size() );
```

18. What is displayed?



3 6 12

```
ArrayList<Integer> a = new ArrayList<Integer>();
for ( int n = 3; n < 20; n = 2*n )
        a.add( n );

for ( Integer i : a )
        System.out.print( i + " ");
```

19. Write an interface named Floatable that has one method: boolean floats()

```
public interface Floatable {
    public boolean floats();
}
```

20. Assume that you have a Boat class that implements Floatable. In the space below, create an array of 10 Floatable objects. Fill the array with 10 Boats (assume that its constructor has no formal parameters).

```
Floatable[] boats = new Floatable[10];
for (int i=0; i < boats.length; i++)
    boats[i] = new Boat();
```

21. Correct the second line so that the boat's move method can be called. This can be done using one or two statements.

```
((Boat) x).move();
```

```
Floatable x = new Boat();
x.move();   // error here

/* Boat has a move method – no
arguments and returns a void */
```

28. Assume that *x* contains the following values:
["0", "0", "4", "2", "5", "0", "3", "0"]
What will the list contain as a after the method is run?

a) ["0", "0", "4", "2", "5", "0", "3", "0"]
b) ["4", "2", "5", "3"]
c) ["0", "0", "0", "0", "4", "2", "5", "3"]
d) ["3", "5", "2", "4", "0", "0", "0", "0"]
e) ["0", "4", "2", "5", "3"]

```
public void  mx(List<String> x){
    int k = 0;
    String ans = "0";
    while (k < x.size() ) {
        if (x.get(k).equals(ans) )
            x.remove(k);

        k++;
    }
}
```

29. Why does this method not compile?

right side must say ArrayList

```
public List<Integer>  mx( int n ){
    List<Integer> x = new  List<Integer>();
    return x;
}
```

Note. LinkedList is another class that implements the List interface.

30. If the array contains { 2, 3, 4 } what does the returned list contain? (in order)

{ 4, 3, 2 }

```
public List<Integer>  mx( int [] a ){
    List<Integer> x = new  LinkedList<Integer>();
    for ( int k = 0; k < a.length; k++ )
        x.add( 0, a[k] );

    return x;
}
```

k: 0
x => 2
4 3

31. Write a method that returns true if a list of integers is sorted in decreasing order. For example, if the list contains { 10, 9, 9, 8 } it should return false because each succeeding value is not less than the previous value. On the other hand, if the list is { 10, 7, 5 } then the method should return true. Return true if the list has less than two elements.

```
public boolean check( List<Integer> x ) {
                                        // assume true, 1 false disproves
int prev = x.get(0);
int i = 1;
while ( i < x.size() ) {
    if ( x.get(i) >= prev )
        return false
    prev = x.get(i)
    i++;
}
return true;
```

22. Select the TRUE statement(s).
a) The first statement creates one array object.
b) The first statement does not create any Radical objects.
c) Radicals may be an abstract class.
d) Radicals may be an interface.
	Radicals [] c = new Radicals[2];
	c[0] = new Joe();
	c[1] = new Moe();

23. Select the TRUE statement. a) PDQ must be a subclass of XYZ. b) XYZ must be an interface. c) PDQ cannot be an abstract class. d) XYZ must be an abstract class. 24. Select the FALSE statement. a) PDQ cannot be an interface. b) XYZ might be a subclass of PDQ. c) XYZ might be an interface. d) XYZ might be an abstract class.	/* Assume the code compiles and runs. */ XYZ betty = new PDQ();

25. What is displayed?
 	
S T P
	ArrayList<String> list = new ArrayList<String>();
	list.add(0, "P"); S R P
	list.add(0, "R");
	list.add(0, "S"); S T P
	list.set(1, "T");
	for (String s : list)
	System.out.println(s);

26. What is displayed?
 	
A B E F
	ArrayList<String> list = new ArrayList<String>();
	list.add("A"); A B C
	list.add("B");
	list.add("C");
	list.remove(2); A B E F
	list.add(2, "E");
	list.add("F");
	for (String s : list)
	System.out.println(s);

27. Select the TRUE statement(s).
 	
a) If Jumbo implements the List interface then this compiles and runs.
b) If Jumbo is the superclass of a class that implements the List interface then this compiles and runs.
c) If Jumbo is the subclass of a class that implements the List interface then this compiles and runs.
	import java.util.*;
	
	public class Runner{
	public static void main(String [] args){
	Jumbo<String> v = new Jumbo<String>();
	mx(v);
	}
	
	public static void mx(List<String> x){
	x.add("hey");
	}
	}