

Ch. 13 Interfaces and Abstract Classes Test

MULTIPLE CHOICE

This question refers to this interface declaration.

```
interface List<E>
{
    public int Size();
    // returns the number of elements in list

    public boolean add(E obj);
    // appends obj to the end of list; returns true

    public void add(int index, E obj);
    // inserts obj at position index (0 <= index <= size),
    // moving elements to the right (adds 1 to their indices) and adjusts size

    public E get(int index);
    // returns element at position index

    public E set(int index, E obj);
    // replaces the element at position index with obj
    // returns the element formerly at the specified position

    public E remove(int index);
    // removes element from position index, moving elements
    // at position index+1 and higher to the left
    // (subtracts 1 from their indices) and adjusts size
    // returns the element formerly at the specified position
}
```



A

1. This interface declaration uses


- a. polymorphism.
- b. inheritance.
- c. generics.
- d. composition.
- e. All of the above

Answer: C

Points: 0 / 1

This question refers to the following interface:

```
interface Qwerty
{
    public void qwertyA();
    public void qwertyB();
    public void qwertyC();
    public void qwertyD();
    public void qwertyE();
    public void qwertyF();
    public void qwertyG();
    public void qwertyH();
}
```

-  B 2. The special abstract class that facilitates using an interfaces, when only a few methods are implemented is called a(n)
- a. empty abstract class
 - b. adapter class
 - c. concrete adapter class
 - d. empty class
 - e. abstract bridge class.

Answer: B

Points: 1 / 1

-  C 3. The Qwerty interface contains 8 practical methods for various utility purposes.
Your new program only requires only the use of method qwertyA.
Which of the following concrete classes properly uses the Qwerty interface?
- a. class Q31 implements Qwerty
- ```

{
 public void qwertyA()
 {
 /* assume method qwertyA is implemented */
 }

 public void qwertyB() { }
 public void qwertyC() { }
 public void qwertyD() { }
 public void qwertyE() { }
 public void qwertyF() { }
 public void qwertyG() { }
 public void qwertyH() { }
}

```
- b. class Q31 implements Qwerty
- ```

{
    public void qwertyA()
    {
        /* assume method qwertyA is implemented */
    }

    public abstract void qwertyB() { }
    public abstract void qwertyC() { }
    public abstract void qwertyD() { }
    public abstract void qwertyE() { }
    public abstract void qwertyF() { }
    public abstract void qwertyG() { }
    public abstract void qwertyH() { }
}

```
- c. class Q31 implements Qwerty
- ```

{
 public void qwertyA()
 {
 /* assume method qwertyA is implemented */
 }
}

```
- d. class Q31 extends Qwerty
- ```

{
    public void qwertyA()
    {
        /* assume method qwertyA is implemented */
    }

    public void qwertyB() { }
    public void qwertyC() { }
    public void qwertyD() { }
    public void qwertyE() { }
    public void qwertyF() { }
    public void qwertyG() { }
    public void qwertyH() { }
}

```

Answer: A

Points: 0 / 1



B

4. Java's Collection is a(n)

- a. subclass.
- b. interface.
- c. concrete class.
- d. subinterface.
- e. abstract class.

Answer: B


Points: 1 / 1

This question refers to this travel program:

Traveling, like many things in life, resemble the abstractness and concreteness of programming.

For instance, on an international trip you may need the following:

- Arrange flights
- Purchase train tickets
- Get documentation ready, like passports and visas
- Exchange dollars into local currency
- Reserve hotels
- Book excursions at different sites of interest
- Learns a few phrases in language of the countries you are visiting

-  A 5. Concrete classes must be created for each of the Western European Countries. This is not complete, we now have an abstract class that already has some common methods finished. Only one concrete class will be shown here.

Which of the following is a concrete class that will properly implement the abstract methods?

```
a. class Italy extends WestEurope
    {
        private String clientName;

        public Italy (String clientName)
        {
            clientName = clientName;
        }

        public void flights()
        { /* assume method is implemented */ }

        public void hotels()
        { /* assume method is implemented */ }

        public void excursions()
        { /* assume method is implemented */ }

        public void phrases()
        { /* assume method is implemented */ }
    }

b. class Italy implements WestEurope
    {
        private String clientName;

        public Italy (String clientName)
        {
            this.clientName = clientName;
        }

        public void flights()
        { /* assume method is implemented */ }

        public void hotels()
        { /* assume method is implemented */ }

        public void excursions()
        { /* assume method is implemented */ }

        public void phrases()
        { /* assume method is implemented */ }
    }
```

```
c. class Italy extends WestEurope
    {
        private String clientName;

        public Italy (String clientName)
        {
            this.clientName = clientName;
        }

        public void flights()
        { /* assume method is implemented */ }

        public void hotels()
        { /* assume method is implemented */ }

        public void excursions()
        { /* assume method is implemented */ }

        public void phrases()
        { /* assume method is implemented */ }
    }
d. class Italy
    {
        private String clientName;

        public Italy (String clientName)
        {
            this.clientName = clientName;
        }

        public void flights()
        { /* assume method is implemented */ }

        public void hotels()
        { /* assume method is implemented */ }

        public void excursions()
        { /* assume method is implemented */ }

        public void phrases()
        { /* assume method is implemented */ }
    }
```

Answer: C
Points: 0 / 1



D

6. Assume that Qwerty is a generic class in Java.
What is true about the following statement:

```
Qwerty<String> q = new Qwerty<String>();
```

- a. It will compile and execute without problems.
- b. Object q will accept any data type.
- c. It will not compile.
- d. Object q will only accept String objects.

Answer: D

Points: 1 / 1

This question refers to this partial program:

```
public class Q1314
{
    public static void main (String args[])
    {
        MyBank tom = new MyBank(5000.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.makeCheckingDeposit(1500.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.makeCheckingWithdrawal(2500.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
    }
}

abstract interface Bank
{
    public abstract double getCheckingBalance();
    public abstract void makeCheckingDeposit(double amount);
    public abstract void makeCheckingWithdrawal(double amount);
}
```



B

7. If the MyBank class implements the Bank interface with these methods, will the program compile?

```
class MyBank implements Bank
{
    private double checking;
    public MyBank(double c)
    public double getCheckingBalance()
    public void makeCheckingDeposit(double amount)
}
{ checking = c; }
{ return checking; }
{checking += amount;}
```

a. No

b. Yes

Answer: A

Points: 0 / 1

E

8. Java's LinkedList is a(n)

- a. interface.
- b. subinterface.
- c. abstract class.
- d. subclass.
- e. concrete class.

Answer: E**Points:** 1 / 1D

9. What is the E in a class heading, like public class List<E> ?

- a. It is a class identifier that can be used with inheritance.
- b. It is a class identifier that can be used with implementing an interface.
- c. It stands for Element.
- d. All of the above

Answer: D**Points:** 1 / 1

This question refers to this program:


```
public class Q1516
{
    public static void main (String args[])
    {
        BankAccounts tom = new BankAccounts(4000.0,6000.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.makeCheckingDeposit(1000.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.makeCheckingWithdrawal(2000.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        System.out.println();
        System.out.println("Tom's savings balance: " + tom.getSavingsBalance());
        tom.makeSavingsDeposit(3000.0);
        System.out.println("Tom's savings balance: " + tom.getSavingsBalance());
        tom.makeSavingsWithdrawal(4000.0);
        System.out.println("Tom's savings balance: " + tom.getSavingsBalance());
    }
}

abstract interface Checking
{
    public abstract double getCheckingBalance();
    public abstract void makeCheckingDeposit(double amount);
    public abstract void makeCheckingWithdrawal(double amount);
}

abstract interface Savings
{
    public abstract double getSavingsBalance();
    public abstract void makeSavingsDeposit(double amount);
    public abstract void makeSavingsWithdrawal(double amount);
}

class BankAccounts implements Checking, Savings
{
    private double checking;
    private double savings;

    public BankAccounts(double c, double s){ checking = c; savings = s; }
    public double getCheckingBalance() {return checking;}
    public double getSavingsBalance() {return savings;}
    public void makeCheckingDeposit(double amount) {checking += amount;}
    public void makeSavingsDeposit(double amount) {savings += amount;}
    public void makeCheckingWithdrawal(double amount) {checking -= amount;}
    public void makeSavingsWithdrawal(double amount) {savings -= amount;}
}
```

-  C 10. You want to add a method called `transferToChecking` that will transfer a specific amount of money from the savings account to the checking account. Which of the following methods is the BEST implementation for this method?

- a.

```
public void transferToChecking(double amount)
{
    checking -= amount;
    savings += amount;
}
```
- b.

```
public void transferToChecking(double amount)
{
    checking += amount;
    savings -= amount;
}
```
- c.

```
public void transferToChecking(double amount)
{
    if (savings > amount)
    {
        checking += amount;
        savings -= amount;
    }
}
```
- d.

```
public void transferToChecking(double amount)
{
    if (savings >= amount)
    {
        checking += amount;
        savings -= amount;
    }
    else
        System.out.println("Insufficient Funds");
}
```

Answer: D

Points: 0 / 1

This question refers to this program:

```

public class Q1720
{
    public static void main (String args[])
    {
        MyBank tom = new MyBank(7000.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.makeCheckingDeposit(2500.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.makeCheckingWithdrawal(1500.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.computeInterest();
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
    }
}


abstract interface Bank
{
    public final double rate = 0.05;
    public abstract double getCheckingBalance();
    public abstract void makeCheckingDeposit(double amount);
    public abstract void makeCheckingWithdrawal(double amount);
    public abstract void computeInterest();
}

class MyBank implements Bank
{
    private double checking;
    private double interest;

    public MyBank(double c)
    {
        checking = c;
        interest = 0.0;
        //rate = 0.1;
    }

    public double getCheckingBalance() {return checking;}
    public void makeCheckingDeposit(double amount){checking += amount;}
    public void makeCheckingWithdrawal(double amount){checking -= amount;}
    public void computeInterest()
    {
        interest = checking * rate;
        checking += interest;
    }
}

```

-  A 11. Will the program above compile if the keyword final is removed from the Bank interface and the comment symbol (//) is removed from the MyBank constructor?
- a. No b. Yes


Answer: A

Points: 1 / 1

- B 12. Will the program above compile as is?
- a. No b. Yes


Answer: B

Points: 1 / 1

-  C 13. A(n) _____ is a group of objects.
- a. class
 - b. interface
 - c. collection
 - d. set
 - e. list

Answer: C

Points: 1 / 1

-  C 14. Which of the following can contain abstract methods?
- I. concrete classes
 - II. interfaces
 - III. abstract classes
- a. II and III only
 - b. I only
 - c. I, II and III
 - d. II only
 - e. III only

Answer: A


Points: 0 / 1

This question refers to this program:

```
public class Q2526
{
    public static void main (String args[])
    {
        MyBank tom = new MyBank(5000.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.makeCheckingDeposit(2500.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.makeCheckingWithdrawal(1500.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
    }
}


abstract class Bank
{
    public abstract double getCheckingBalance();
    public abstract void makeCheckingDeposit(double amount);
    public abstract void makeCheckingWithdrawal(double amount);
}

class MyBank extends Bank
{
    private double checking;
    public MyBank(double c) {checking = c;}
    public double getCheckingBalance() {return checking;}
    public void makeCheckingDeposit(double amount){checking += amount;}
    public void makeCheckingWithdrawal(double amount){checking -= amount;}
}
```

-  D 15. What would be the result of changing the keyword `extends` to `implements` in the definition of `MyBank`?
- The program will compile, execute, and crash immediately.
 - The program will not compile.
 - The program will execute, display different output from the previous program, and then crash.
 - The program will execute, display the same output as the previous program, and then crash.
 - The output would be the same as the previous question.

Answer: B

Points: 0 / 1

-  D 16. The reserved word `abstract` - in an abstract class declaration - is
- required for the class heading and for every method in the class.
 - required for every abstract method in the class.
 - required for adapter classes only.
 - optional, just like an interface.
 - required for the class heading only.


Answer: B

Points: 0 / 1

This question refers to these declarations:


```
abstract class Bank1
{
    public abstract double getCheckingBalance();
    public abstract void makeCheckingDeposit(double amount);
    public abstract void makeCheckingWithdrawal(double amount);
}

abstract interface Bank2
{
    public abstract double getCheckingBalance();
    public abstract void makeCheckingDeposit(double amount);
    public abstract void makeCheckingWithdrawal(double amount);
}
```

-  C 17. Syntactically, are Bank1 and Bank2 the same, except for the words interface and class?
- a. No, because an abstract class always must extend an interface.
 - b. No, because the abstract class declaration does not require the abstract reserved words.
 - c. Yes they are.
 - d. No, because the interface declaration does not require the abstract reserved words.
 - e. No, because an abstract class always must implement an interface.

Answer: D

Points: 0 / 1

-  E 18. Abstract classes
- a. are required for polymorphism.
 - b. are used with composition only.
 - c. involve both implementation and inheritance.
 - d. must implement all interface methods.
 - e. require abstract methods only.


Answer: C

Points: 0 / 1

This question refers to this partial program:

```
public class Q1112
{
    public static void main (String args[])
    {
        MyBank tom = new MyBank(3000.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.makeCheckingDeposit(1000.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
        tom.makeCheckingWithdrawal(5000.0);
        System.out.println("Tom's checking balance: " + tom.getCheckingBalance());
    }
}

interface Bank
{
    public double getCheckingBalance();
    public void makeCheckingDeposit(double amount);
    public void makeCheckingWithdrawal(double amount);
}
```

-  B 19. What is the output of this program, if the MyBank class implements the Bank interface in this way?

```
class MyBank implements Bank
{
    private double checking;
    public MyBank(double c) { checking = c; }
    public double getCheckingBalance() { return checking; }
    public void makeCheckingDeposit(double amount){ checking -= amount; }
    public void makeCheckingWithdrawal(double amount){checking += amount;}
}

a. Tom's checking balance: 3000
   Tom's checking balance: 2000
   Tom's checking balance: 7000
b. Tom's checking balance: 3000
   Tom's checking balance: 4000
   Tom's checking balance: -1000
c. Tom's checking balance: 3000
   Tom's checking balance: 4000
   Tom's checking balance: 9000
d. Tom's checking balance: 3000
   Tom's checking balance: 4000
   Tom's checking balance: 4000
```

Answer: A

Points: 0 / 1



A

20. A(n) _____ is a linear collection that allows access to any element.
Duplicates may exist.

- a. list
- b. interface
- c. collection
- d. class
- e. set

Answer: A

Points: 1 / 1