Note: In the following questions, the phrase "**could be replaced**" means that the resulting program would be equivalent to the original one, i.e., result in the same behavior.

Answer A for True, B for false for each question 1 through 11 *(1 pt. each):*

1. Line 4 contains an assignment (B)
2. In line 4, an object is created (B)
3. In line 7, answer could be replaced by this.answer (B)
4. Line 12 is in the scope of adjustAnswer (A)
5. Line 12 contains a declaration (B)
6. Line 12 is in the scope of avg (B)
7. Line 12 is in the scope of newAnswer (A)
8. In line 12, this.answer could be replaced by answer (A)
9. Line 19 is in the scope of avg (B)
10. In line 19, someAnswer is an instance variable (B)
11. In line 23, the value of avg changes by losing any fractional part it may have (B)
12. *(2 pts.)* Suppose you are told `"answer < newAnswer"` . This is an example of
    A. a pre-condition of the setAnswer method
    B. a post-condition of the setAnswer method
    C. a class invariant

```
             public class MidtermQuestion {

                public static final int MAXVAL = 100;
line  4 ....... private int answer;

                public MidtermQuestion(int answer) {
line  7 ........... setAnswer(answer);
                }

                public void setAnswer(int newAnswer) {
                   if (newAnswer > MAXVAL) {
line 12 .............. this.answer = MAXVAL;
                   }
                   else {
                      this.answer = newAnswer;
                   }
                }

                public void adjustAnswer(double someAnswer) {
line 19 ........... if (someAnswer > MAXVAL) {
                      return;
                   }
                   double avg = (this.answer + someAnswer) / 2.0;
line 23 ........... this.answer = (int) avg;
                }
             }
```

13. *(3 pts.)* Given the statements below,

```
boolean found = false;
boolean ok = true;
double width = 12.4;
```

what is the value of

```
(found || ok) && (width >= (int) width)
```

A. 12
B. 12.4
C. 13
D. false
E. true

14. *(3 pts.)* Suppose that the Java statements in the problem above are followed by this additional statement:

```
found = (x == ((int)width < height));
```

Assume it compiles and executes without error. What can you conclude about the type of the variable x that appears in that expression?

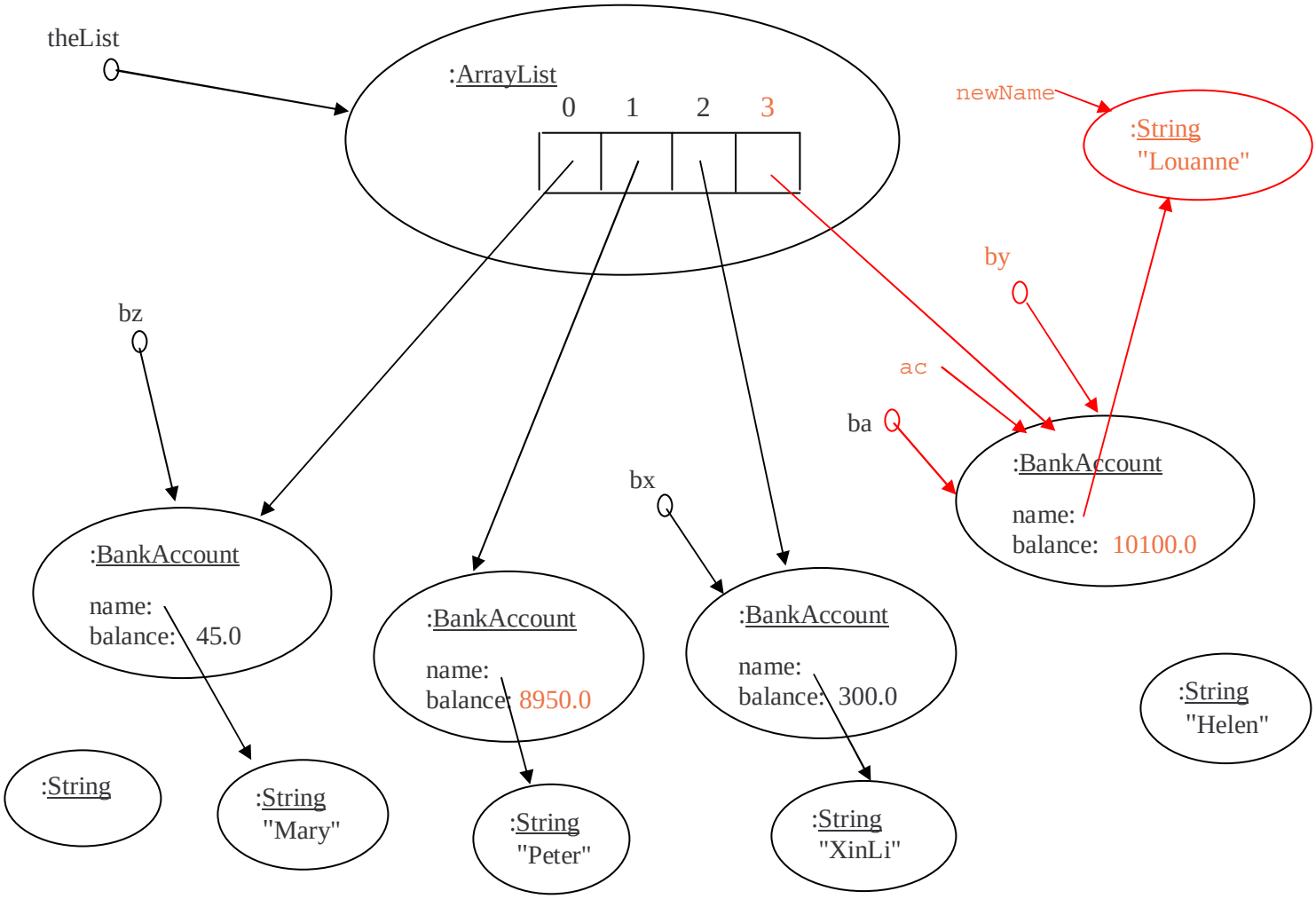
A. x is an int
B. x is a boolean
C. x is a double
D. can't determine type of x without knowing the type of height

15. *(3 pts.)* Is there an error in trying to add num20 to the list?

```
ArrayList bList = new ArrayList();
int num20;
bList.add(num20);
```

A. compile error -- an int cannot be added to an ArrayList
B. run-time error -- an int cannot be added to an ArrayList
C. num20 cannot be added to the ArrayList because num20 has not been initialized.
D. There is no error.

Suppose a series of statements has executed, resulting in the situation pictured in the diagram:



16. *(8 pts.)*

(a). Write a short sequence of Java statements (at most four statements) which deposits 100.0 dollars in Helen's account and withdraws 50.0 dollars from Peter's account.

```
by.deposit(100.0);
ac = (BankAccount)theList.get(1);
ac.withdraw(50.0);
```

(b). Update the picture above to reflect the changes made by your code. (Do not redraw the whole diagram!)

17. *(5 pts.)* Suppose the following statements then execute.
Update the diagram to show their effect.

```
BankAccount ba = by;
String newName = "Louanne";
by.changeName(newName);        // sets instance variable name to value of parameter
theList.add(by);
```

Recall the MovieRecord class which stores information on an individual movie and its methods from assignment 4:

```
getTitle     that returns the movie title, a String type
getDirector  that returns the movie director, a String type
getYear      that returns the year the movie was made, an int type
getActor1    that returns the first prominent actor, a String type
getActor2    that returns the second actor, a String type
getActor3    that returns the third actor, a String type
toString     that returns all the movie data as one String object
```

and the MovieReporter class which manages a collection of MovieRecord objects.

18. *(14 pts.)* Write a void method for your MovieReporter class, called printInDirector, that takes a String as a parameter and prints all the MovieRecords (using toString) that have that exact String in the director's name. For example, if the parameter value is "Ford", since "Ford" is a substring of the directors "Ford", "John Fordham", "Ford Frick", and so on, those are printed. **You must use an Iterator.** Assume the ArrayList instance variable of MovieReporter is called **mvList**.

Hint: The following method of String may be of use:

```
/** indexOf   If the String argument str occurs as a substring within this
 *  object, then the index of the first character of the first such substring
 *  is returned; if it does not occur as a substring, -1 is returned.
 */
public int indexOf(String str) { ... }


public void printInTitle(String str){

      Iterator iter = theList.iterator();

      while(iter.hasNext()){

            MovieRecord currentRecord = (MovieRecord)iter.next()
            if( curtentRecord.getDirector().indexOf(str) != -1){
                  System.out.println(currentRecord);
            }
      }
}
```

19. *(21 pts.)* Write a method with a boolean return type for your MovieReporter class, called hasBusyDirectors, that takes an ArrayList of MovieRecords as a parameter and returns whether or not any director of a MovieRecord in the list is a "busy director." Being a busy director means that that director made at least two movies in the list.

Do not use the Java *break* statement and **do not use Iterators**. The only methods of ArrayList you are allowed to **use are *get* and *size***. While correctness of your solution is most important, a full credit solution will also be straightforward, concise, and follow good programming practice.

```
boolean hasDuplicates(ArrayList list){

      for(int i = 0 ; i < list.length() ; i++){
            for(int j = j ; j < list.length() ; j++){
                  if( ((MovieRecord)list.get(i)).getDirector().equals(
                        ((MovieRecord)list.get(j)).getDirector() ) ){
                        return true;
                  }
            }
      }
      return false;
}
```