





# Exam B – Solutions

Exam A has the same multiple choice questions, but in different order.

2. **TRUE** (a) or **FALSE** (b): A superclass object will never have access to more non-private methods than its subclasses.
3. When the `main` method is written in a UML class diagram, it will appear as:
- (a) `+main(args: String[]): void`
  - (b) `-main(args: String[]): void`
  - (c) `+main(args: String[]): void`
  - (d) `+main(): void`
  - (e) none of the above
4. Which one of the following can the word `final` NOT be used with:
- (a) a property
  - (b) a method
  - (c) a class
  - (d) an abstract method
  - (e) none of the above
- A method cannot be both abstract and final*
5. Given an array loaded using `int[] Ar = {2,4,6,8,10};` which one of the following statements successfully prints out each element of the array *counting down* from the largest value to the smallest?
- (a) `for(int x=Ar.length; x=0; x--) System.out.println(Ar[x-1]);`
  - (b) `for(int x=Ar.length; x>0; x++) System.out.println(Ar[x-1]);`
  - (c) `for(int x=Ar.length; x > 0; x--) System.out.println(Ar[x-1]);`
  - (d) `for (int x=Ar.length; x > -1; x--) System.out.println(Ar[x-1]);`
  - (e) none of the above

6. **TRUE** (a) or **FALSE** (b): A no-arg constructor is *not* an essential feature of a superclass if that superclass has no other constructors. It is only when that superclass has *any* constructors that a no-arg constructor becomes necessary.
7. **TRUE** (a) or **FALSE** (b): You cannot instantiate a class that contains an abstract member.
8. **TRUE** (a) or **FALSE** (b): When using file IO, the term 'input' is used to indicate that data is taken *from* a program and input *into* a file.
9. **TRUE** (a) or **FALSE** (b): The abstract members of a superclass must always be overridden in its first non-abstract subclass.
10. If a method is declared without an access modifier, then, assuming it is in a public class, its visibility is limited to  
(a) the class    (b) the package    (c) the project    (d) subclasses    (e) none of the above
11. **TRUE** (a) or **FALSE** (b): A UML diagram can only describe *is a* and *has a* relationships between classes, nothing else.
12. Assume that Classroom and Room are two classes directly related by inheritance. Given what you know about the relative complexity of superclasses and subclasses, which one of the following is most likely to be correct?  
(a) public class Room extends Classroom  
(b) public class Classroom extends Room

13. Which one of the following symbols is used to indicate that a method is protected in UML?
- (a) -                      (b) +                      (c) @                      (d) #                      (e) none of the above
14. TRUE (a) or FALSE (b): Subclasses inherit their superclass constructors
15. Which one of the following symbols is used to indicate inheritance between a superclass and its subclass:
- (a)  (b)  (c)  (d)  (e) none of the above
16. In Java File IO, the “try-with-resources” feature ensures that
- (a) you must use the resources in the block of code that follows
- (b) it is associated with error handling
- (c) this is a special File IO method that opens both text-based and binary files
- (d) default File IO resources will be employed in accessing the file
- (e) none of the above
17. Assume the following class

```
public class TestEven{  
    public static boolean isEven(int number){  
        return (number%2==0);  
    }  
}
```

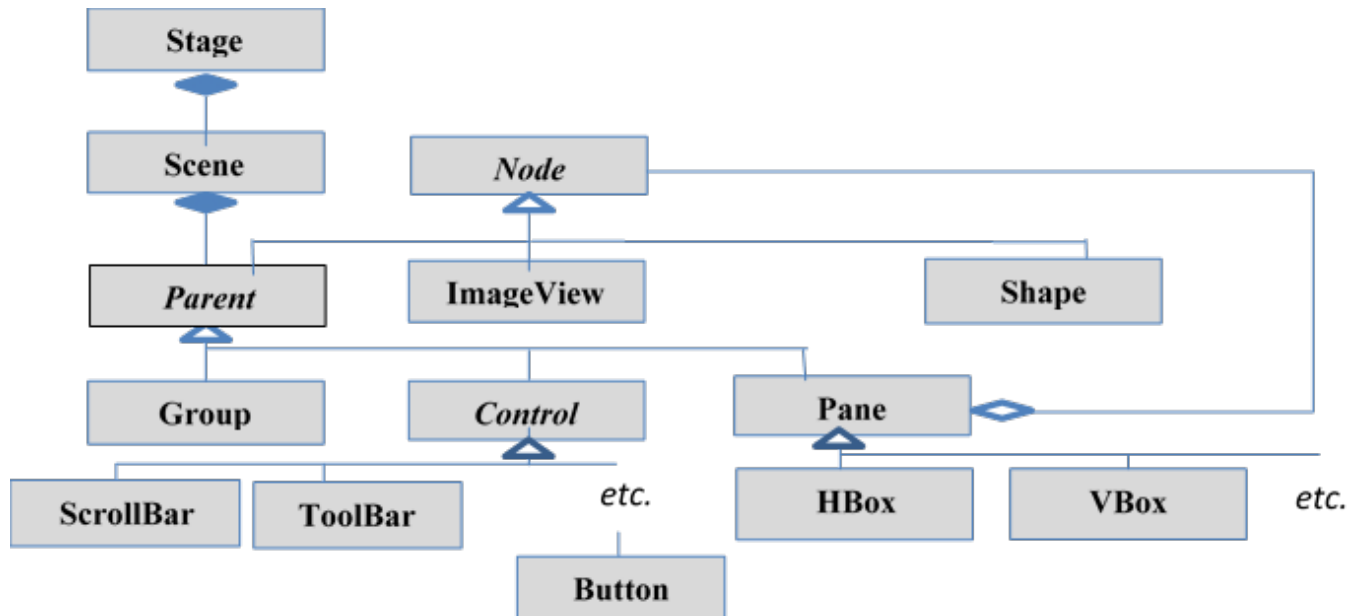
// con't next slide

17. (con't) Which one of the following JUnit methods would you probably use to test `TestEven.isEven(3)` ?

- (a) `assertBoolean()`      (b) `assertTrue()`      (c) `assertFalse()`  
(d) `assertEquals()`      (e) none of the above

18. TRUE (a) or FALSE (b): Every class has a superclass

*Note: for the questions below, you may find the following JavaFX UML diagram useful*



19. Which one of the following `Application` methods starts the execution of a JavaFX application, before all the others?  
(a) `launch()`      (b) `start()`      (c) `init()`      (d) `stop()`      (e) none of the above
20. **TRUE** (a) or **FALSE** (b): Any variable that holds a `Stage` object should never be declared `final`.  
*There's nothing wrong with making a variable final if it isn't going to be changed during execution. And since you typically load the stage only once, it's probably a good idea to make it final*
21. Which one of the following classes supplies methods to its subclasses designed to handle collections of "children"?  
(a) `node`      (b) `control`      (c) `pane`      (d) `parent`      (e) none of the above
22. **TRUE** (a) or **FALSE** (b): Graphical applications in JavaFX are *event driven*

- 23.** If you attempt to load and display a new `Control` object in the `primaryStage` using the code

```
Control myCtl = new Control();  
Scene myNewScene = new Scene(myCtl, 500, 300);  
primaryStage.setScene(myNewScene);  
primaryStage.show();
```

the result will be :

- ☒ (a) An error on the first line of the code fragment above
  - ☐ (b) An error on the second line of the code fragment above
  - ☐ (c) The window will display correctly, but the `Control` object will be stretched to fit the scene
  - ☐ (d) The window will not display anything
  - ☐ (e) none of the above
- 24.** Which one of the following JavaFX objects is a control whose contents cannot be changed by the user (i.e. during program execution) by first inserting the cursor, and then typing a new string into the object itself?
- (a) `Text`      (b) `TextField`      (c) `TextArea`      ☒ (d) `Label`      (e) none of the above

**Part B: Terminology – worth 1 mark each**

FILL IN THE SINGLE BEST ANSWER—ONE WORD OR SYMBOL IN THE EACH SPACE PROVIDED BELOW.

**USE ONE WORD ONLY IN EACH SPACE; DO NOT USE ACRONYMS**

26. The keyword final, when used in a class declaration, indicates that a superclass cannot be extended to a subclass

27. The Software Development Life Cycle describes the various stages in the evolution of a software project, from its inception through to finished product (3 words)

28. A(n) stream can be thought of as data ‘flowing’ through computer memory, which a program can read from, even when new data is being written to it.

29. An abstract class is one that can’t be instantiated into an object; it must first be extended to a(n) concrete class (Note: “non-abstract”, or any of its variants, is not a correct answer)

30. *is a* and *has a* relationships are more formally described by the terms inheritance and composition, respectively.

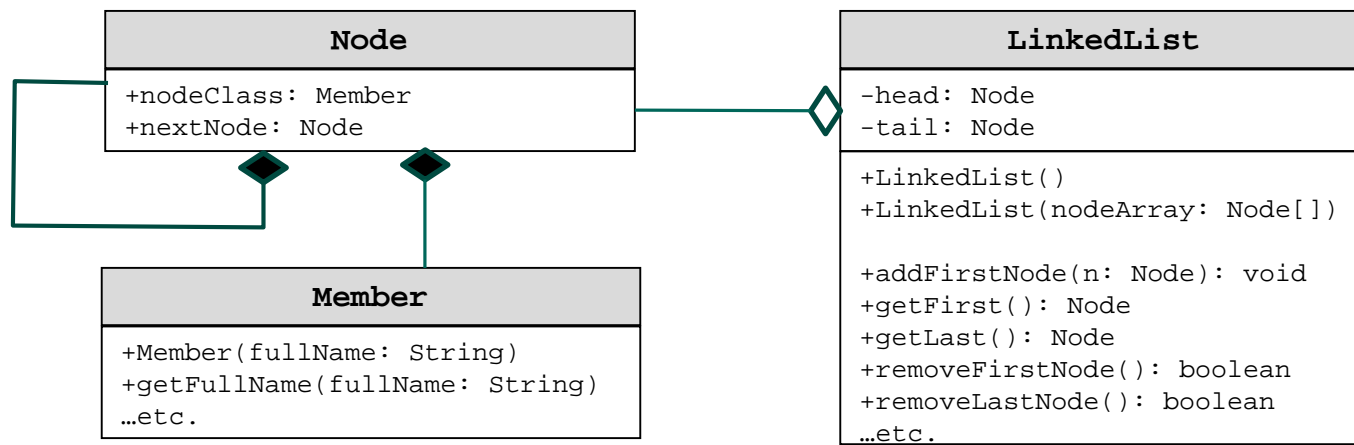
**31.** Give three short, distinct reasons why it is preferable to chain constructors and methods.  
2 Marks ea.

- a) *Ease of Maintenance:* Changing the output—say "Hello" to "Hi" in the `SayHello` class—doesn't require that you change every constructor, just the final one.
- a) *Code Reuse:* Once you have one of your methods working, why reinvent the wheel again and again? Simple chain to it;
- b) *Less Chance to Introduce New Errors during Upgrades:* when changes are needed, there is less code to alter, hence less chance of introducing mistakes. You only need to upgrade one constructor in the chain, not all of them;
- c) *Consistent Usage:* by chaining your constructors, all constructors behave in a predictable fashion, because they are all, ultimately, based upon the same base constructor. The client of such a class understands this implicitly, and doesn't have to relearn each constructor separately, since they can be relied upon to all behave in the same, intuitive manner.

**32.** A *Linked List* is a data structure consisting of a set of *nodes* chained together sequentially. (Note: this *node* is not to be confused with a `Node` object used in JavaFX. In the case, a *node* is any class that acts as a single link in a sequential chain of other nodes. This has nothing to do with the graphical `Node` class.) Each node holds, in addition to a reference to an object, the next node in the list. The list is transversed by starting at the first node and using it to locate the second node. The second node contains a link to the third, the third to the fourth, and so on to the end.

Shown at right below is the `LinkedList` class. It holds the first node in the list (the “head”) and the last (the “tail”), as well as methods that allow the user to add, get, and remove nodes. The `Node` on the left stores both a reference to the next node in the chain, as well a member object. (So a linked list may be thought of, by analogy, as a necklace: the nodes are the links in the chain, the member objects are jewels connected to each link in the chain, and the `LinkedList` is a clasp that ties the two ends of the chain together).

Assume we wish to store a list of club members using a linked list. The class used, called `Member`, is indicated below left. Using standard UML symbols (i.e. lines, arrows and diamonds) show how these three class would be connected to one another.



**33.** In the box below, explain why it is not strictly necessary to use `@Override` when overriding abstract methods, even when a method's signature in the subclass does not match the abstract method's signature in the superclass.

An abstract method *must* be overridden in its (concrete) subclass(es). Therefore, if the signature is wrong in the subclass, the JVM will flag an error, even if `@Override` is *not* used. So it is not necessary to use `@Override` when overriding *abstract* methods when the subclass signature is wrong, since the JVM will catch the error anyway.

*Note: a number of students apparently failed to read the entire question, explaining that `@Override` was not necessary if the method signature was correct, but only essential if the signature was incorrect, and that the `@Override` was a convenience, a signature-checker, etc., thus missing the fact that this is a question about overriding abstract superclass methods in the subclass, and not about using `@Override` per se.*

*Read the entire question before answering; don't assume you know what the question is about in advance.*

**34.** The code on the next three pages contain no fewer than 17 errors, which could be manifested as either compile-time warnings or run-time errors. The latter may be due to logic errors on the part of the programmer. Your job is to find 12 of those errors. **Place a number from 1 to 12 next to each of these errors, circle each of these numbers, and explain why the code is in error in the box on page 9, next to the number you just circled in the code.**

**Description:** The following program\* consists of five classes, highlighted in bold. A **Shape** class stores the `x` and `y` coordinates of a Shape object. It contains an abstract method `getArea()`. Additionally, it contains the `inputAllFields()` method, that prompts the user to input the `x` and `y` coordinates. **Circle** and **Rectangle** are two classes that extend Shape. They each override `getArea()` with an appropriate calculation. Additionally, they override `inputAllFields()` and `toString()` in the superclass and append their own information on to these methods. The **Input** class contains two static methods that prompt the user for input, display a string output, and then checks to see that the input is between two appropriate ranges before prompting the user for input; otherwise, the user is prompted to re-enter the values. Finally, the **TestShapes** class runs the `main()` routine, which prompts the user to input some number of shapes, and prompts the user for the coordinates and measurements of each shape, before finally calculating each result. Note that for simplicity, the only shapes used are `Circles` and `Rectangles`.

\*From code originally supplied by Rex Woolard

```
public class TestShapes {
```

```
    public static void main(String[] args) {  
        private Shape[] listShapes = new Shape[10];
```

```
        int numShapes = Input.getInt("Input number of shapes (10 max):", 1, 10);
```

```
        for (int i = 1; i < numShapes; i++) {
```

```
            Shape shape;
```

```
            if (Input.getInt("\nChoose 1:Circle 2:Rectang
```

```
                shape = new Circle();
```

```
            else
```

```
                shape = new Rectangle();
```

```
            shape.inputAllFields();
```

```
            listShapes[i] = shape;
```

```
        }
```

```
        System.out.print("\nCalculating Area of Shapes\n");
```

```
        for (Shape shape : Shapes[])
```

```
            System.out.println(shape);
```

```
    }
```

① Difference between actual output and code output ('10' v. '20')

①

② for loop starts at 1, therefore i=0 will not be loaded

③ Output will include null in all the uninitialized elements of memory, unless you limit the for loop to numShapes elements

④ Must use listShapes in *enhanced for*, not Shapes[] or Shape[] which is a declaration only

```

public class Shape {
    private final static double MAX_X = 1920.0, MAX_Y = 1080.0;
    private double x, y;

    public Shape() {}
    public abstract double calcArea(){};
    public void inputAllFields() {
        x = input.getDouble("Enter x:", 0.0, MAX_X);
        y = input.getDouble("Enter y:", 0.0, MAX_Y);
    }

    @Override
    public String toString() {
        return((this instanceof Circle)?"Circle":"Rectangle" +
            " coordinates are (" + x + ", " + y + "), area = " + calcArea());
    }
}

```

⑤ class has abstract method, hence it must be abstract

⑥ must be static final, not final static

⑦ calcArea is abstract, hence there should be no body, including anything after ()

⑧ getDouble() is a static method of class Input, hence should be Input, not input

⑨ this refers to Shape, and a Shape is not an instance of a Circle. Hence the *conditional if* always returns the String "Rectangle"

⑩ instanceof should be instanceof

```

public class Circle extends Shape {
    private static final double MIN_RADIUS = 1.0;
    private static final double MAX_RADIUS = 1000;
    private double radius;

    @Override
    public void inputAllFields() {
        super.inputAllFields();
        radius = Input.getDouble("Enter radius:",
                                MAX_RADIUS, MIN_RADIUS);
    }

    @Override
    public double calcArea() {
        return Math.Pi * radius * radius;
    }

    @Override
    public String toString() {
        return(super.toString() + "radius: " + radius);
    }
}

```

11 MAX\_RADIUS and MIN\_RADIUS reversed

12 Should be PI not Pi

```

public abstract class Rectangle extends Shape {
    private static final double MIN_DIM = 1.0;
    private static final double MAX_DIM = 1000;
    private double length, width;

    @Override
    public void inputAllFields() {
        super.inputAllFields();
        length = Input.getDouble("Enter " +
                                "length:", MIN_DIM, MAX_DIM);
        width = Input.getDouble("Enter "
                                "width:", MIN_DIM, MAX_DIM);
    }

    @Override
    public double calcArea() {
        return super.x * super.y;
    }
    13
    public String toString() {
    14 System.out.println(super.toString() + "
        length: " + length + " width: " + width);
    }
}

```

13 calculation is wrong;  
should be length \* width  
(x and y are coordinates of  
the Shape object)

14 should return String, not  
print it out

```
import java.util.Scanner;
```

```
public class Input {
```

```
    private Scanner input = new Scanner(System.in);
```

```
    private static double d; private static int i;
```

```
    public static double getDouble(String s, double min, double max) {
```

```
        System.out.print(s + "(" + min + "-" + max + "): ");
```

```
        do{
```

```
            d = input.nextInt();
```

```
            input.nextLine();
```

```
        } while ((d < min) || (d > max));
```

```
        return d;
```

```
    }
```

```
    public static int getInt(String s, int min, int max){
```

```
        System.out.print(s + "(" + min + "-" + max + "): ");
```

```
        do{
```

```
            i = input.nextInt();
```

```
            input.nextLine();
```

```
        } while ((i < min) || (i > max));
```

```
        return i;
```

```
    }
```

```
}
```

15 min and max are backward from output string

16 Needs to use nextDouble() to read in a double value d

17 missing ; after while

18 Scanner needs to be declared static if input is to be referenced inside static method

Note: in keeping with previous midterms, the last three entries in this question were treated as bonus questions. Hence your exam was marked out of 55, not 58