

Exam A – Solutions

- Exams B and C have same questions, but in different order.
 - Note that the wording of questions may be slightly different between different versions of the exam (i.e. the answer is true on some tests and false on others), so look closely at the question before appealing your mark. If the question was ‘flipped’, the answer will be also.
2. TRUE (a) or FALSE (b) Every member of a superclass is available in any of its subclasses
 3. When one constructor is tied to another, more-powerful constructor by use of `this()`, it is said to be:
(a) overloaded (b) overridden (c) chained (d) static (e) none of the above
 4. Which one of the following statements is **NOT TRUE** of the *enhanced for* loop
(a) Unlike the regular *for* loop, it uses the colon (:), not the semicolon (;) in its format
(b) You can only count up with an *enhanced for*; you cannot count down.
(c) You can get data *out* of an array using the *enhanced for*; you cannot set the data into an array using the *enhanced for*.
(d) You cannot break out of the *enhanced for* prematurely, you must go through every element in the array first
(e) none of the above

5. Which one of the following methods must *first* be executed to begin a JavaFX application?
(a) `init()` (b) `start()` (c) `launch()` (d) `stage()` (e) none of the above
6. Which one of the following best describes a variable that holds a reference value in Java (for example the word `input` in `Scanner input;`)
(a) It holds the actual address of an object
(b) Once declared (as above), it automatically holds the reference value of an object.
(c) It can hold the reference value for objects of the named class, or objects of any subclass.
(d) Once the reference variable is loaded, it cannot be changed.
(e) None of the above.
7. Which one of the following statements is **TRUE**: The definition
`ArrayList<String> months = new ArrayList<>();`
(a) instantiates an array object named `months` that is no different from an array defined as
`String[] months = new String[];`
(b) flags an error, since the generic type is unspecified on the right hand side of the definition
(c) instantiates a special kind of array, one that initially holds a single `String` element, by default
(d) uses the identifier `months` to hold an ID that points to an array of `Strings`
(e) none of the above

Note: I would also accept (d) as the correct answer, since the wording is unclear as to what the type of the array object is held in `months`

8. 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-1; x=0; x--) System.out.println(Ar[x]);`
- (b) `for(int x=Ar.length-1; x>0; x++) System.out.println(Ar[x]);`
- (c) `for(int x=Ar.length-1; x > 0; x--)`
`System.out.println(Ar[x]);`
- ☒ (d) `for (int x=Ar.length-1; x > -1; x--)`
`System.out.println(Ar[x]);`
- (e) none of the above

9. **TRUE** ☒ (a) or **FALSE** (b): you cannot inherit from a final class

10. Which one of the following is **NOT** a reference type in Java?

- (a) an array (b) a class (c) an interface (d) a String ☒ (e) none of the above

11. **TRUE** ☒ (a) or **FALSE** (b): arrays are mutable objects

12. **TRUE** ☒ (a) or **FALSE** (b): The second (or middle) compartment of a UML class diagram may be omitted if the fields of that class are private

13. Assume array1 and array2 are each arrays of integers having equal length. Which one of the following is NOT a proper way to copy the contents of array2 into array1?





- (a) array1 = array2;
- (b) array1 = array2.clone();
- (c) for (int j = 0; j < array2.length; j++){
 array1[j] = array2[j];}
- (d) int j=0;
 for (int thisInt: array2){array1[j++] = thisInt;}
- (e) none of the above

14. **TRUE** (a) or **FALSE** (b): you cannot define one class within another.

15. When public static void main(String[] args) 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

16. **TRUE** (a) or **FALSE** (b): It is not absolutely necessary to use @Override when overriding abstract methods, since signature differences between the superclass and subclass will be detected automatically by Eclipse.

17. Which one of the following classes contains static methods suitable for sorting, say, an array of strings?
- (a) the Array class
 - ☒ (b) the Arrays class
 - (c) the ArrayList class
 - (d) the primitive array class used by the Java language has its own sorting method built-in
 - (e) none of the above
18. Which one of the following is **NOT TRUE** of a class which contains an abstract method:
- ☒ (a) All the methods in the class will be treated as abstract
 - (b) The class cannot be instantiated
 - (c) The abstract method must be overridden in a subclass, unless the subclass is itself abstract
 - (d) The class can be inherited by a concrete class
 - (e) None of the above
19. Which one of the following symbols would be used to indicate that one class has many other classes (i.e 'has a' relationship involving one-to-many classes):
- (a) 
 - (b) 
 - ☒ (c) 
 - (d) 
 - (e) none of the above
20. **TRUE** (a) or **FALSE** (b): mutable variables can potentially lead to deadlock
21. **TRUE** (a) or **FALSE** (b): ArrayLists can only be used with reference types, not primitive types

22. Which abstract JavaFX class contains the methods required to organize the 'children' in its subclasses?
(a) Node (b) Control (c) Pane (d) Parent (e) none of the above

23. TRUE (a) or FALSE (b): An inner class can only have a single method.

24. Which one of the following statements best describes what the `copyOf` method does?

```
+copyOf(original: T[], newLength: int): <T> T[]
```

`copyOf` is a public method that

- (a) copies an array of data type `original` and returns `newLength` values of type `original`
- (b) copies an array of data type `T` and returns `newLength` values of type `T`
- (c) copies `newLength` elements of the array `T[]` back to the same array
- (d) returns `newLength` elements of whatever kind of array is passed as the first parameter, as specified in the `<T>` notation
- (e) none of the above

25. In the `copyOf` declaration in the previous question, `T` is referred to as the
(a) general type (b) generic type (c) class type (d) parameterized type
(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. A stream can be thought of as a sequence of data flowing through memory.
27. After an object has been instantiated, setting its value to null invokes garbage collection
28. A copy constructor is a special kind of constructor that takes an instance of an object as an input parameter and creates a new reproduction of it.
29. polymorphism is a fundamental OOP feature that allows a variable of a superclass type to be used to store or pass a subclass object
30. A lambda expression uses the `->` notation to specify what code will be executed when a particular `ActionEvent` occurs.
31. Java's 4 levels of access protection include `private`, `protected`, `public` and package

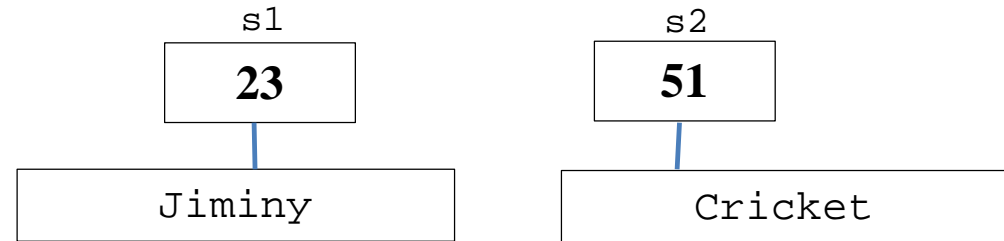
32. Consider the code at right. (Line number are given so that they can be referred to in the question below. They would not, of course, appear in the actual code.)

```
1. public class Question31 {  
2.     public static void main(  
           String[ ] args) {  
3.         String s1 = "Jiminy ";  
4.         String s2 = "Cricket";  
5.         s1 = s1 + s2;  
6.         s2 = s1;  
7.         s1 = "Pinocchio";  
8.         System.out.println(s2);  
9.     }  
10. }
```

Assume the boxes below hold the reference values for `s1` and `s2`, along with the strings they point to, immediately *after* lines 4 and 5 have been executed. In the remaining boxes, fill in the reference values and the strings they point to (there are six boxes to fill in). Note that if a new reference value would need to be created after a particular line was executed, then you should make up a new numeric value, rather than use an existing number, to indicate this, and insert this value into the appropriate box.

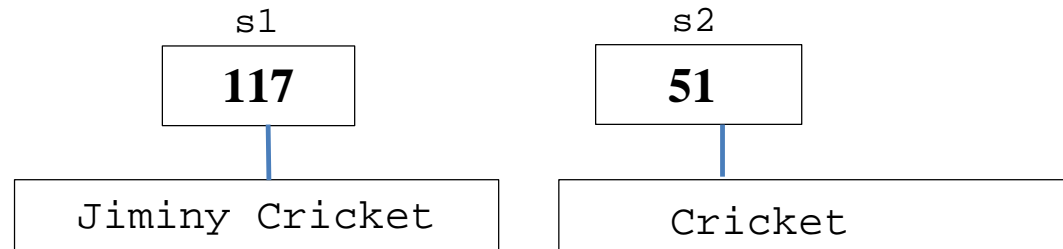
(a) Fill in the blanks below with appropriate reference values and strings

After line 4:



After line 5:

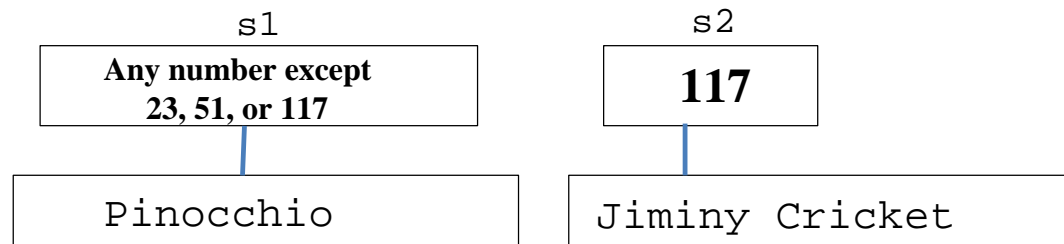
5. `s1 = s1 + s2;`



After line 7:

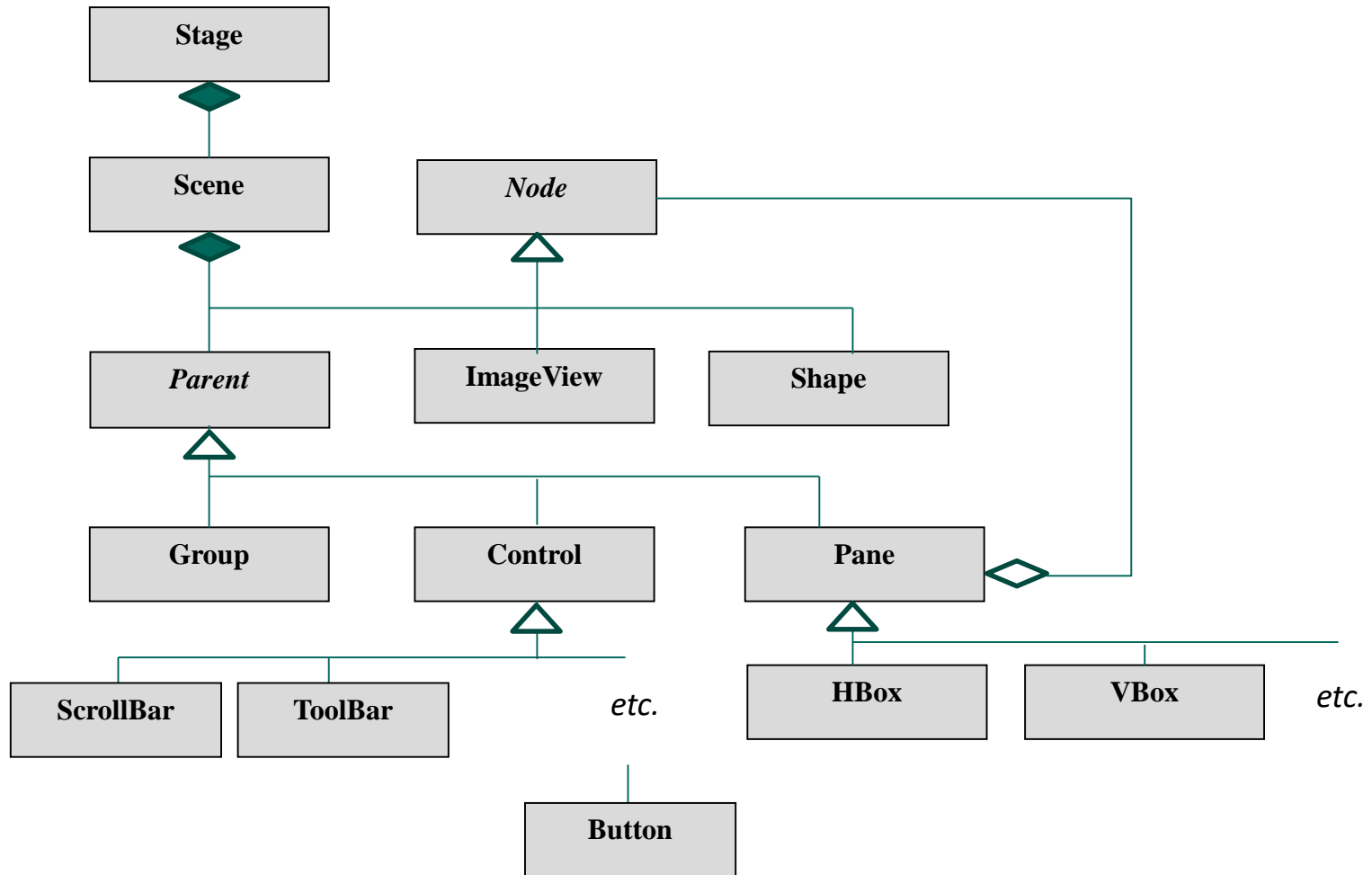
6. `s2 = s1;`

7. `s1 = "Pinocchio";`



(b) Explain why the value of `s1` changes after line 5 is executed. Why is the reference value in `s1` not the same as it was after line 4? Does the string “Jiminy Cricket” (after line 5) start at the same location in memory as “Jiminy” (after line 4)?

Strings are immutable, which means that, once created, they remain in memory until garbage collected (or the program ends). So when we execute `s1 = s1 + s2`, we do not simply ‘add’ one string to another in memory; a new string is created, with a new reference value in `s1`, at a new location. Hence ‘Jiminy’ and ‘Jiminy Cricket’ refer to two separate strings, at two different locations.



Given the JavaFX UML class diagram shown above, explain the following:

(a) Can you use a new `ScrollBar` object as shown in the code below, given the information provided in the UML diagram above? Why or why not? Give a brief explanation using *is a* and *has a* arguments

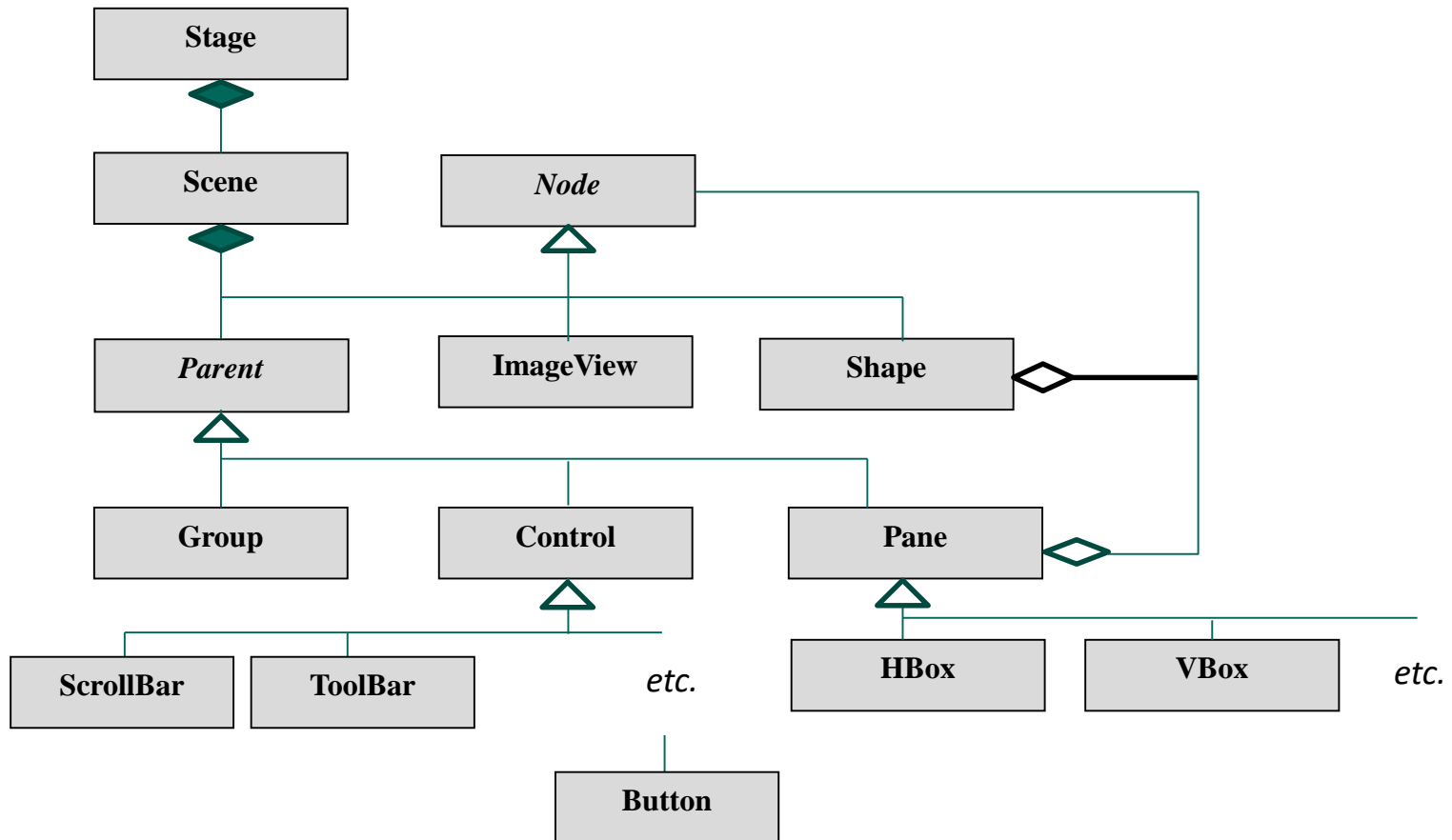
```
ScrollBar scrollbar = new ScrollBar();  
Scene scene = new Scene(scrollbar, 450, 200);  
primaryStage.setScene(scene);  
primaryStage.show();
```

Yes. From the diagram above, a Scene has a Parent. Since a ScrollBar is a Control, and a Control is a Parent, therefore any ScrollBar object is a Parent, and can be used to satisfy the requirement that a Scene has a Parent, as seen in the second line of code above.

(b) Why can an `HBox` hold another `HBox`? Provide a short answer based on the UML diagram above using *is a* and *has a* arguments:

An HBox is a Pane, a special kind of object designed to hold other Node objects. This is shown in the diagram: A Pane 'has a' many Nodes. Node is an abstract class, but via polymorphism any subclass of a Node can also be stored in a Pane. This includes Shapes, Scrollbars, Toolbars, and even other Panes, including other HBox's. Thus, because an HBox is a Node, and Pane/HBox has a many Nodes, an HBox can store other HBoxes.

(c) Indicate, in the diagram below, what connection would need to exist to allow a Shape object to hold other Node objects. In other words, if you were responsible for designing JavaFX, and you wanted to indicate that Shapes could hold other Groups, Controls and Panes—even other Shapes—what one simple connection would you need to add to the above diagram? Indicate this by amending the diagram above with the appropriate UML symbol(s).



34. What is the difference between a `final` method and an `abstract` method? Can a method be both `final` and `abstract`? Explain why or why not below

No. A `final` method is one which *cannot* be overwritten. 'Final' means 'constant', which is used to indicate that something cannot be changed. An `abstract` method is one that *must* be overridden in some concrete subclass. Therefore, a `final` method cannot be an `abstract` method.



35. The code on the next three pages contain no fewer than 20 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

```

import java.util.ArrayList;
import java.util.Scanner;
public class TestShapes {
    public static void main(String[] args) {

        ArrayList<Shape> listShapes = new ArrayList();
        int numShapes = Input.getInt("Input number of shapes:", 1, 10)

        for (int i = 0; i < numShapes; ++i) {
            Shape shape;
            if (Input.getInt(
                "\nChoose 1:Circle  2:Rectangle", 1, 2) == 1)
                shape = new Circle();
            else
                shape = new Rectangle();
            shape.inputAllFields();
            listShapes[i] = shape;
        }

        System.out.println("All shapes found? " + (i==numShapes))
        System.out.print("Calculating Area of Shapes\n");

        for (Rectangle rect : listShapes)
            System.out.println(rect);
    }
}

```

① ArrayList missing diamond brackets after new ArrayList; should be new ArrayList<>();

② Missing ; at end of line

③ listShapes is an arrayList; it uses .add, not [i]

④ cannot use i==numShapes; i out of scope outside for

⑤ must use Shape shape, not Rectangle rect, since a Rectangle cannot hold a circle object

```

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

    public abstract double calcArea; ⑥
    public void inputAllFields() {
        x = input.getDouble("Enter x coordinate:", 0.0, MAX_X);
        y = input.getDouble("Enter y coordinate:", 0.0, MAX_Y);
    } ⑦

    @Override
    public String toString() {
        System.out.println((this instanceof Circle)? ⑨
            "Circle" ⑧ "Rectangle" + ⑩
            " coordinates are (" + x, + y + "), area = " + calcArea(); ⑪
    }
}

```

⑥ calcArea needs () after identifier

⑦ Should be Input, not input

⑧ prints String, but needs to return String

⑨ instanceof not instanceOf

⑩ conditional if uses : not ;

⑪ needs + " , " +

⑫ missing final)

```

public class Circle extends Shape {
    private static final double MIN_RAD=1.0;
    private static final double MAX_RAD=1000;

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

    @Override 13
    public abstract double calcArea() {
        return 2* Math.PI * radius; 14
    }

    @Override
    public String toString() {
        return(super.toString() + "radius: " +
            radius);
    } 15
}

```

¹³ calcArea declared as abstract in superclass; needs to be concrete in subclass

¹⁴ Calculates perimeter, not area

¹⁵ radius declared in inputAllFields but is only available there. The declaration for radius needs to be at the top of the class, so the radius will have global scope inside the entire class

```

public class Rectangle extends Shape {
    private static final double MIN_DIMEN = 1.0;
    private static final double MAX_DIMEN= 1000;

    private double length, width;

    @Override
    public void inputAllFields() {
        16 inputAllFields();
        length = Input.getDouble("Enter length:",
            MIN_DIMEN, MAX_DIMEN);
        width = Input.getDouble("Enter width:",
            MIN_DIMEN, MAX_DIMEN);
    }

    @Override
    public double calcArea() {return x * y;} 17

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

```

16 Needs super, otherwise this is used and the code calls itself forever

17 Supposed to return area, instead it multiplies the x and y coordinates; Should be length* width

```

public class Input { 18 19
    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.nextDouble();
            input.nextLine();
        } while ((d > min) || (d < max)) 20
        return d; 21
    }

    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;
    }
}

```

18 Missing `import.util.Scanner;`
(It appear in `TestScanner`, where it isn't used; that declaration needs to be here)

19 `input` needs to be static, otherwise you can't save the double value returned to `d`.

20 Missing `;`

21 Logic is backwards; you should keep looping while the input value is *less than* the minimum allowed or *greater than* the maximum allowed. Either change `> <` to `< >`, or change `||` to `&&`