

1. Which one of the following methods is *first* executed to begin a JavaFX application?  
(a) `init()`    (b) `start()`    (c) `launch()`    (d) `stage()`    (e) none of the above
2. **TRUE** (a) or **FALSE** (b): you cannot extend a final class
3. Which one of the following is **NOT TRUE** of a class that contains an abstract method?
  - a. The class cannot be instantiated
  - b. The abstract method must be overridden in a subclass, unless that subclass is also abstract
  - (c) All the methods in the class will be treated as abstract
  - d. The class can be inherited by any concrete subclass
  - e. None of the above
4. **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
5. 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
6. In a UML diagram, static methods are identified by being:  
a) underlined    b) *italicized*    c) **bold-face**    d) CAPITALIZED    e) none of the above

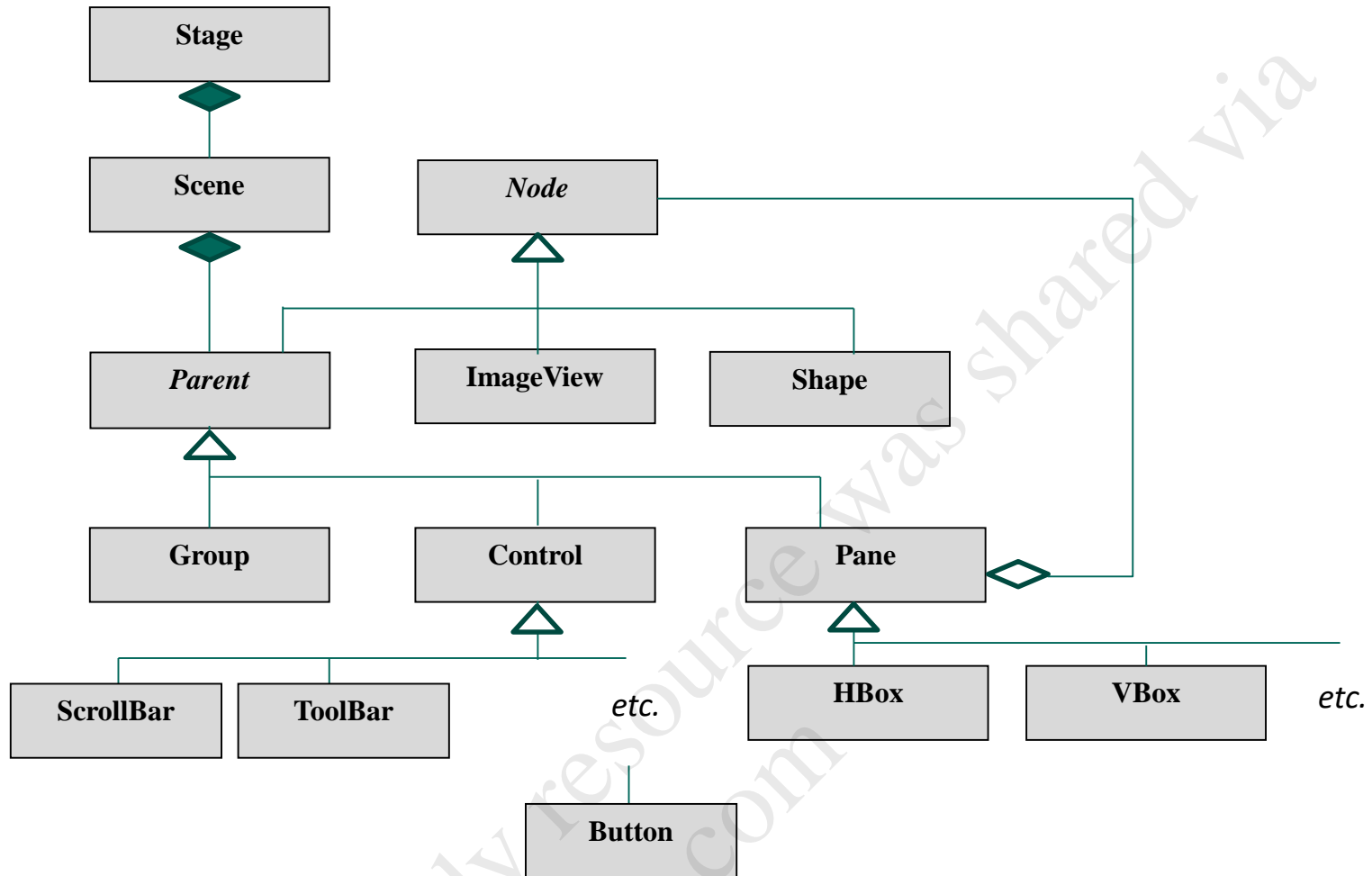
7. Which one of the following words is **NOT** used to indicate a relationship between objects?  
a) association    **b) aggression**    c) composition    d) inheritance    e) none of the above
8. **True** (a) or **False** **(b)**: the return type of a method is part of its signature
9. Which one of the following does **NOT** require a Java *identifier*?  
a) a method    b) a property    **c) a keyword**    d) a class    e) none of the above
10. Which one of the following JavaFX objects is used to structure the contents of a Scene?  
a) A Stage    b) a Parent    c) an ObservableList  
**d) a Pane**    e) none of the above
11. **True** (a) or **False** **(b)**: To create a JavaFX application, the programmer must first instantiate a new `Application` object, and only when the application is fully loaded and ready for display, call `Application.show()`.
12. `@Override` has the effect of:  
(a) forcing methods in the superclass to be overridden by methods in the subclass  
(b) checking that the spelling of the subclass method/constructor name is correct  
**(c) checking that the method and its parameters on the line that follows actually exists in the superclass**  
(d) adding a new method to the subclass which does not exist in the superclass  
(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**

13. A stream can be thought of as a sequence of data flowing through memory.
14. When an object is declared, its value is set to null by default
15. A copy constructor is a special kind of constructor that takes an instance of an object as an input parameter reproduces an identical object from it.
16. polymorphism is a fundamental OOP feature that allows a variable of a superclass type to be used to store or pass a subclass object
17. Java's 4 levels of access protection include `private`, `protected`, `public` and package



18. 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.*

## 19. In file President.java:

```
public abstract class President {  
    private boolean POTUS 1  
    private String favSong, presName;  
    public void President 2 (String prezName, String s){  
        favSong = (s == "" 3) ? "Hail to the Chief": s;  
        presName = prezName;  
    } 4  
    public boolean isPOTUS(){return POTUS;}  
    public abstract void setPOTUS (boolean result);  
}
```

1. Missing ; (given)
2. President constructor returns a void
3. Should be s==" for comparison; 's=' is an assignment
4. President needs a *no-arg* constructor, since it is being used as a superclass, and already has a two-string constructor

## In file Candidate.java:

```
public class Candidate extends President{  
    private double percentOfVote; 5  
    private String realMotto = "Vote for me";  
    private String name;  
    private String victorySong;  
  
    public Candidate(String name){  
        this(name, "The Winner Takes It All");  
    }  
  
    public Candidate(String name, String songTitle){  
        this.setName(name);  
        this.setVictorySong(songTitle);  
    } 6  
  
    public Candidate(String name){  
        if (isPOTUS())  
            7 super(setName(), "I Did It My Way");  
    } 8  
  
    public double getPercent(){return pctOfVote;} 9  
    public double setPercent(double vote%){  
        10 vote% = pctOfVote; 11  
    } 12  
  
    public String getRealMotto() {return realMotto;}  
    public void setRealMotto(String rm){realMotto = rm;}  
    public void setName(String name){this.name= name;} 13  
    public void setVictorySong(String song){  
        this.victorySong = song;  
    } 14  
    public void setPOTUS(){setPOTUS("true");} 15  
} 16
```

5. Should be pctOfVote to be consistent with usage elsewhere
6. Candidate has two constructors with the same signature
7. super() must be in first line
8. Need to pass name to President constructor, hence requires a getter, not a setter
9. Needs ; inside braces, not outside
10. Setters return void, not double
11. Cannot use % in variable name
12. vote% = pctOfVote is backwards
13. realMotto spelled wrong, needs two 't's
14. setPOTUS() must have @Override, or the signature must match the method in the abstract class, otherwise it does not correctly override the abstract method in the superclass
15. setPOTUS() must take a boolean value, hence it should be passed a true or false, but not a String
16. setPOTUS() calls on itself, hence forms an infinite loop

## In file TestCandidate.java

```
public class TestCandidate {  
    public static void main(String args){  
        float result = 0.54;  
        Candidate dt = new Candidate("Trump");  
        dt.setRealMotto("Double down on  
            stupidity. Vote Trump");  
  
        Candidate hc= new Candidate("Clinton");  
        hc.setRealMotto("Untrustworthy, but  
            better than Trump. Vote Clinton");  
  
        hc.setPercent(result);  
        dt.setPercent(1 - result);  
  
        if (hc.getPercent() > dt.getPercent())  
            dt.setPOTUS;  
        else  
            hc.setPOTUS;  
    }  
}
```

- 17. Needs [], i.e. String[] args
- 18. result should be double, not float, since double is the data type that gets passed to setPercent()
- 19. Should be <, not >, otherwise the loser wins
- 20. setPOTUS is a method, hence it should be setPOTUS()
- 21. Missing closing brace