1. The following code snippets are legal.

   (a) `ArrayList<String> a = new LinkedList<String>();`
   True / False

   (b) `List<Integer> l = new List<Integer>();`
   True / False

   (c) `Set<Double> s = new TreeSet<Double>();`
   True / False

2. What is the difference between a class and an object?

3. What does the `super` keyword do? When must you use it?

4. When overriding equals() for a custom object, what other method should you also override and why? Do you need to use accessors and mutators in either of these methods to access the class' private variables and methods?

5. Identify the errors in the following code:

```
1 public interface TestInterface {
2     public TestInterface() {}
3     public void splat( String message ) {}
4 }
5 public class AbstractClass {
6     abstract protected void myMethod();
7     public void callMyMethod() {
8         myMethod();
9     }
10 }
11 public class ConcreteClass extend AbstractClass implement TestInterface {
12     protected int myMethod() {
13         System.out.println( "I got called!" );
14         return 7;
15     }
16     public void splat( String message ) {
17         System.out.println( "SPLAT!" );
18         System.out.println( message );
19     }
20 }
21 public class SimpleProgram {
22     public static void main(String[] args) {
23         AbstractClass myclass = new AbstractClass();
24         myclass.callMyMethod();
25         myclass.splat( "woohoo" );
26     }
27 }
```

6. What are the two type of exceptions? What is an Error?

7. What will be printed by the following code?

```
1 public class ExceptionTest {
2     public static void main( String args[] ) {
3         try {
4             for ( int i = 0; i < 5; i++ ) {
5                 process( i );
6             }
7         } catch ( Exception e ) {
8             System.err.println( e.getMessage() );
9         }finally{
10            System.out.println( "Done processing" );
11        }
12    }
13    public static void process( int num ) throws Exception {
14        if ( num != 2 ) {
15            System.out.println( "Good number: " + num );
16        } else {
17            throw new Exception( "Bad number: " + num );
18        }
19    }
20 }
```

8. Instantiate a new `BufferedReader` which decorates a `ForumReader`. ForumReaders only use the default constructor but can throw a `TooLongDidntReadException` if the forum post is too long to be worth reading. Handle any exceptions thrown and gracefully close the file.

9. Why is buffering a good idea when doing file I/O?

10. Name and describe 3 JavaFX layouts.

11. What is the default behavior of a button in JavaFX? What other events can be triggered?

12. Write an `EventHandler` implementation called `SimpleAction` that changes a label('messageLabel') to say "Cheese!" when an `MouseEvent` is received.

13. Draw the tree of the hierarchy of the layouts used in the following code.

```
1  public class BorderPaneDemo extends Application {
2
3      /**
4       * start constructs the application layout using a BorderPane.
5       * @param stage container in which the UI will be rendered
6       */
7      public void start( Stage stage ) {
8
9          BorderPane border = new BorderPane();
10
11         // left side
12         border.setLeft( this.makeButtonList() );
13
14         // top side
15         FlowPane top = new FlowPane();
16         TextField text = new TextField( "Text 'in the top'" );
17         // have to align the TextField (not the flow)
18         text.setAlignment( Pos.TOP_CENTER );
19         border.setTop( text );
20
21         // right side
```

```
22          border.setRight( new CheckBox( "Check Me" ));
23
24          // bottom
25          FlowPane bottom = new FlowPane();
26          bottom.setAlignment( Pos.CENTER_RIGHT );
27          bottom.getChildren().add( new Label( "This label is immutable." ) );
28          border.setBottom( bottom );
29
30          // center
31          border.setCenter( this.makeGridPane() );
32
33          stage.setTitle( "JavaFX BorderPaneDemo" );
34          stage.setScene( new Scene( border ) );
35          stage.show();
36      }
37
38      /**
39       * makeButtonList a pane with a 'list' of buttons.
40       * @return pane for inclusion in another pane/scene
41       */
42      private Pane makeButtonList() {
43
44          Pane pane = new VBox();
45          for ( int i = 1; i < 7; ++i ) {
46
47              Button btn = new Button( "Button " + i );
48              pane.getChildren().add( btn );
49          }
50          return pane;
51      }
52
53      /**
54       * makeGridPane creates and returns a grid layout of buttons.
55       * @return grid pane that can be added to a region
56       */
57      private GridPane makeGridPane() {
58
59          GridPane grid = new GridPane();
60
61          for ( int row = 0; row < 5; ++row ) {
62              for ( int col = 0; col < 6; col++ ) {
63                  grid.add(
64                          new Button( "HitButton(" + row + "," + col + ")" ),
65                          col, row );
66              }
67          }
68
69          grid.setGridLinesVisible( true ); // makes it hard to see active item
70          return grid;
71      }
72
73      /**
74       * The main is the application entry point to launch the JavaFX GUI.
75       * @param args not used
76       */
77      public static void main( String[] args ) {
```

```
78          Application.launch( args );
79      }
80
81  }
```

14. Describe `List, Set` and `Map`. What interface(s) do `List` and `Set` extend?

15. Explain the difference between `Comparator` and `Comparable`.

16. Use an iterator to count the number of elements in the Collection `victims`.

```
public int numVictims(Collection<Person> victims) {




}
```

17. Write a `Comparator` that compares strings by their size modulo (%) 13 and use this comparator for sorting a TreeSet.

18. Write a lambda using stream operations to print the names of the dogs whose age is at most 2 and is of the breed Corgi.

```
List<Dog> dogs = Arrays.asList(
    new Dog("Brady", "Corgi", 1),
    new Dog("Bruno", "Pug", 2),
    new Dog("Porter", "Corgi", 2),
    new Dog("Darcy", "Dalmation", 3),
    new Dog("Rodney", "Bulldog", 3)
);
```

19. Describe the following thread functions/terms

    (a) start

    (b) sleep

    (c) join

(d) synchronized

(e) wait

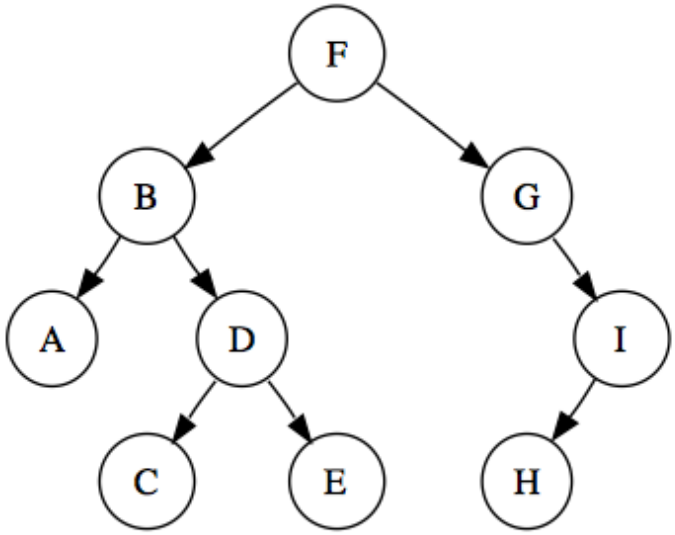
(f) notify

(g) notifyAll

20. Write a program which creates 3 threads to print the numbers 0-999, 1000-1999, 2000-2999 respectively.

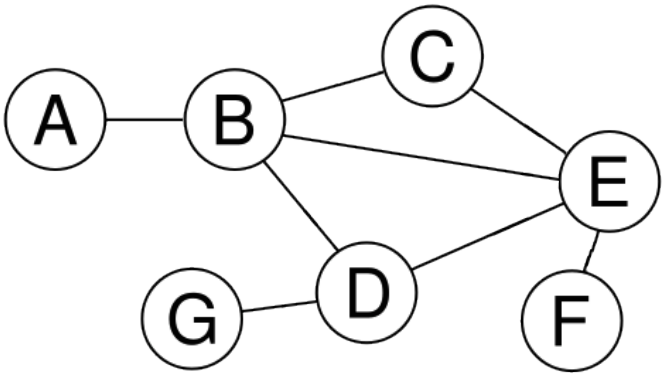
21. For the following two problems, always visit neighbors in alphabetical order.

    (a) Given the following *tree*, perform a DFS from node F to E and give the order of visited nodes.
    Repeat with a BFS.



(b) Given the following *graph*, perform a DFS from node A to F and give the order of visited nodes.

Repeat with a BFS.



22. Given the `searchBFS` function from lecture, trace the output when run between nodes A and F from the previous question (b).

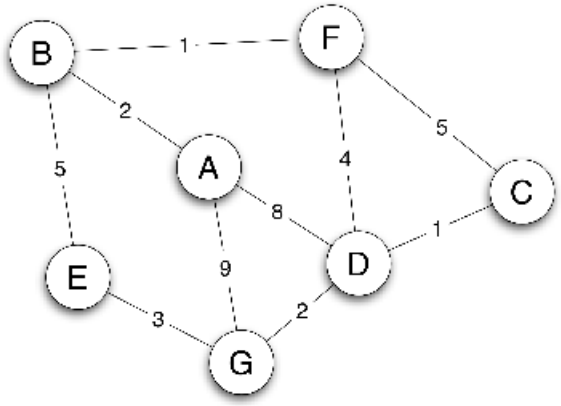
    (a) Show the resulting predecessor map.

    (b) Show how the queue and current change over time.

    (c) What does constructPath() return?

```
1     public List<Node> searchBFS(String start, String finish) {
2             Node startNode, finishNode;
3             startNode = graph.get(start);
4             finishNode = graph.get(finish);
5
6             List<Node> dispenser = new LinkedList<Node>();
7             dispenser.add(startNode);
8
9             Map<Node, Node> predecessors = new HashMap<Node,Node>();
10            predecessors.put(startNode, startNode);
11
12            while (!dispenser.isEmpty()) {
13                Node current = dispenser.remove(0);
14                if (current == finishNode) {
15                    break;
16                }
17                for (Node nbr : current.getNeighbors()) {
18                    if(!predecessors.containsKey(nbr)) {
19                        predecessors.put(nbr, current);
20                        dispenser.add(nbr);
21                    }
22                }
23            }
24
25           return constructPath(predecessors, startNode, finishNode);
26        }
27
28        private List<Node> constructPath(Map<Node,Node> predecessors,
29                Node startNode, Node finishNode) {
30            List<Node> stack = new LinkedList<Node>();
31            List<Node> path = new ArrayList<Node>();
32
33            if(predecessors.containsKey(finishNode)) {
34                Node currNode = finishNode;
35                while (currNode != startNode) {
36                    stack.add(0, currNode);
37                    currNode = predecessors.get(currNode);
38                }
39                stack.add(0, startNode);
40            }
41
42            while (!stack.isEmpty()) {
43                path.add(stack.remove(0));
44            }
45
46            return path;
47        }
```

23. Perform Dijkstra's shortest path algorithm on this graph to find the shortest distance from A to C. In the case of a tie, select a path based on the alphebetical ordering of the vertices.



Iteration	Finalized Vertex	A	B	C	D	E	F	G
		(0,None)	(∞,None)	(∞,None)	(∞,None)	(∞,None)	(∞,None)	(∞,None)
1	A	✓						
2								
3								
4								
5								
6								
7								

24. Give the least cost path from A to E and the total cost.

25. Give the least cost path from A to C, and the total cost.

26. Give the definitions for the following:

    - Serializable

    - Domain Name Server

    - TCP

    - UDP

    - Datagram

27. Complete the following code for the backtracking solve algorithm. Be sure to include pruning in your solution.

```
1     public static ArrayList< Configuration > solve(Configuration config){
2         //Backtracking algorithm
3         if (config.isGoal()){
4             return config;
5         } else {
6             ArrayList< Configuration > neighbors =  config.getSuccessors();
7             for (int i = 0; i < neighbors.size(); i++) {
8
9
10
11
12
13
14
15
16
17            }
18        }
19        return null;
20    }
```

28. `Magic Square:` An n x n square with numbers 1 to n^2 arranged, so that the sum of the rows, the columns, and the diagonals is the same. The figure shows the solution for a 3x3 magic square. What algorithm would you choose to solve the Magic Square Problem? Explain the details.