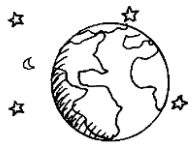
File Output

4.1 K

Name: _____

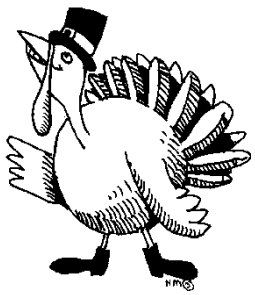
1. Use the information on the left to fill in the code to print the desired data to the files.

In the file named `a.txt`,
Print "12345"
 
	`try {`
	`  FileOutputStream out = openFileOutput("_____", Activity.MODE_PRIVATE);`
	`  out.write(_____);`
	`  out.flush();`
	`  out.close();`
	`} catch (FileNotFoundException e) {`
	`  e.printStackTrace();`
	`} catch (IOException e) {`
	`  e.printStackTrace();`
	`}`

You've decided to name your FileOutputStream `cat`.
 	
In the file named `b.txt`,
Print "5678"
	`try {`
	`  FileOutputStream _____ = openFileOutput("_____", Activity.MODE_PRIVATE);`
	`  _____.write(_____);`
	`  _____.flush();`
	`  _____.close();`
	`} catch (FileNotFoundException e) {`
	`  e.printStackTrace();`
	`} catch (IOException e) {`
	`  e.printStackTrace();`
	`}`

You've decided to name your FileOutputStream `joe`.
 	
In the file named `c.txt`,
Print "23", then "45", then "67"
	`try {`
	`  FileOutputStream _____ = openFileOutput("_____", Activity.MODE_PRIVATE);`
	`  _____.write(_____);`
	`  _____.write(_____);`
	`  _____.write(_____);`
	`  _____.flush();`
	`  _____.close();`
	`} catch (_____ e) {`
	`  e.printStackTrace();`
	`} catch (IOException ___) {`
	`  e._____();`
	`}`

2. Put the lines of code in order. (1 = first)



_____ `turkey.close();`

_____ `turkey.write(33);`

_____ `try {`

_____ `e.printStackTrace();`

_____ `turkey.flush();`

_____ `} catch (FileNotFoundException e) {`

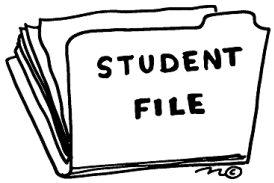
_____ `FileOutputStream turkey = openFileOutput("gobble.txt", Activity.MODE_PRIVATE);`

3. Circle the most correct answer.
   (a) A file is stored in the phone's RAM. This means that it is erased when the program is done running.
   (b) Files are like variables. Both are temporary.
   (c) Variables are temporary, but files are not. They are saved on the SSD on the phone.
   (d) Files are rarely used in programs because no one ever needs to save their work.
   (e) Files cannot be opened and closed when they are needed.

4. Read the code and answer the true/false questions about it.
This code would open a file called data.txt and print the numbers in the array a to it.

```
try {                                                    ← File Output needs a Try/Catch statement.

   FileOutputStream out = openFileOutput("data.txt", Activity.MODE_PRIVATE);
                                                         ← The name of the file.
                                                           If it does not exist, it will be created.
   for(int i=0; i<a.length; i++) {
      out.write(a[i]);                                   ← Outputs an int to the file in one line.
   }

   out.flush();
   out.close();                                          ← Closes the file.

} catch (FileNotFoundException e) {
   e.printStackTrace();
                                                         ← Catches the errors to prevent crashes.
} catch (IOException e) {
   e.printStackTrace();
}
```

T F a) The name of the file was `out.txt`.
T F b) The name of the `FileOutputStream` object was out
T F c) A try/catch is required if a user wished to output to a file.
T F d) `Data.txt` is stored on the Hard Drive.
T F e) Text files can be opened and updated from a program.
T F f) `println` is the command to print to a file.
T F g) If the file doesn't exist when you output to it, java will create it.
T F h) A file would be useful in a chess game to save your progress until the next time you play.
T F i) A file would also be useful to remember information that you need on different screens.



5. Print the following arrays to a file using a loop to save them for the next time a user plays the game.

• Your onClick is `save`.
• Your FileOutputStream is called `yes`.
• Your filename is `a.txt`
• The global array you wish to print out is:
`int scores []= {1,2,3,4,5,6};`
	`public void _____(View view){`
	` try {`
	`  FileOutputStream ____ = openFileOutput("___.txt", Activity.MODE_PRIVATE);`
	`  for(int i=0; i<_____; i++) {`
	`  ____.write(_____);`
	`  }`
	`  ____.flush();`
	`  ____.close();`
	` } catch (FileNotFoundException e) {  e.printStackTrace();`
	` } catch (IOException e) {  e.printStackTrace();`
	` }`
	`}`

• Your onClick is `newGame`.
• Your FileOutputStream is called `file`.
• Your filename is `b.txt`
• The global array you wish to print out is:
`int b[][]= {{1,2,3,4}, (5,6,7,8}};`
 	
`int row=2;`
`int col=4;`
	`public void _____(_____ _____){`
	` try {`
	`  FileOutputStream ____ = openFileOutput("___.txt", Activity.MODE_PRIVATE);`
	
	`  }`
	`  ____.flush();`
	`  ____.close();`
	` } catch (FileNotFoundException e) {  e.printStackTrace();`
	` } catch (IOException e) {  e.printStackTrace();`
	` }`
	`}`

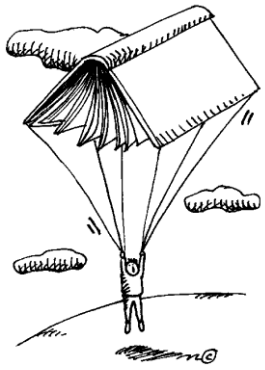
File Input Stream

4.2 K

Name: _____

6. Highlight the lines with the variable "total". This code reads in 12 numbers from a file and totals them.

```
public void open(View view){
    try {
        FileInputStream mInput = openFileInput("data.txt");
        int total = 0;
        for(int i=0; i< 12; i++) {
            total+= = mInput.read();
        }
        mInput.close();
        TextView words = (TextView) findViewById(R.id.words);
        words.setText("The total is " + total);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```



7. Fill in the code to print the desired things in the files.

In the file named "a.txt", Read in one line.	
	`_____ {`
	`  FileInputStream in = openFileInput("_____");`
	`  int data = in.read();`
	`  in.close();`
	`} _____ (FileNotFoundException e) {`
	`  e.printStackTrace();`
	`} _____ (IOException e) {`
	`  e.printStackTrace();`
	`}`

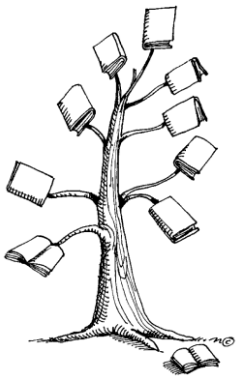
You've decided to name your FileInputStream "james".
 	
In the file named "b.txt", Read in two numbers from the file and print them on the screen.
	
	`try {`
	`  FileInputStream _____ = openFileInput("_____");`
	`  _____ data = _____.read();`
	`  _____ data2 = _____.read();`
	`  _____.close();`
	`  TextView words = (_____) findViewById(R.id.words);`
	`  _____.setText(data + " " + data2);`
	`} catch (FileNotFoundException e) {`
	`  e.printStackTrace();`
	`} catch (_____ e) {`
	`  e.printStackTrace();`
	`}`

You've decided to name your FileInputStream "beth".
 	
In the file named "c.txt", Read in 8 lines and print them to the screen.
	`try {`
	`  FileInputStream _____ = openFileInput("_____");`
	`  String display="";`
	`  for(int i=0; i<8; i++) {`
	`  int data = _____.read();`
	`  display+=data+" ";`
	`  }`
	`  _____.close();`
	`  _____ words = (TextView) findViewById(R.id.words);`
	`  words._____(display);`
	`} catch (FileNotFoundException e) {`
	`  e._____();`
	`} catch (IOException e) {`
	`  e.printStackTrace();`
	`}`

8. Circle the most correct answer.

T F a) FileInputStreams can be closed when you are done with them.
T F b) Reading from and writing to a file are slow operations.
T F c) Integer.parseInt is used with in.read() so that you can do math operations.
T F d) Files persist when an app is closed. They can be used to save progress.
T F e) read() is used to read in Strings that are stored in a file.

9. Put the lines of code in order. (1 = first)



```
_____   try {
_____   e.printStackTrace();
_____   FileInputStream in = openFileInput("data.txt");
_____   a[i] = in.read();
_____   in.close();
_____   catch (FileNotFoundException e)
```

10. Match each line with its description. Write the match in the FIRST column.

	`FileInputStream in`		a) Opens the specified file name.
	`in = openFileInput("data.txt");`		b) Begins a try / catch block.
	`a[i] = in.read();`		c) Ends a try / catch block.
	`for(int i=0; i<a.length; i++) {`		d) Declares a new FileInputStream..
	`try {`		e) Reads in one line from the file.
	`catch (FileNotFoundException e)`		f) Loops for the length of the array.

11. Create programs to read in each of the following arrays.

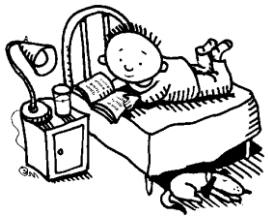
- The filename is `apple.txt`
- The FileInputStream is `input`
- The array to be filled is:

```
int a[]=new int[100];
```

```
try {
  FileInputStream _________ = openFileInput("____________");


  _________.close();
  redraw();
} catch (FileNotFoundException e) {
  e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
}
```

- The filename is `bob.txt`
- The FileInputStream is `bobby`
- The array to be filled is:

```
int row=90;
int col=80;
int b[][]=new int[row][col];
```



```
try {
  FileInputStream _________ = openFileInput("____________");


  _________.close();
  redraw();
} catch (FileNotFoundException e) {
  e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
}
```

Basic Objects 4.3 T

Name: _____

1. Label each method with its type.
(constructor, accessor, mutator, facilitator)

```
public class Animal {
```

Class piece:

```
    private String noise;
    private String type;
```

Method Type:

```
    public Animal () {
        noise = "meow";
        type = "cat";
    }
```

Method Type:

```
    public Animal (String a, String s) {
        noise = s;
        type = a;
    }
```

Method Type:

```
    public String toString () {
        return "the " + type + " says " +
            noise;
    }
```

Method Type:

```
    public String getSound () {
        return noise;
    }
```

Method Type:

```
    public String getAnimal () {
        return type;
    }
```

Method Type:

```
    public void setSound (String s) {
        noise = s;
    }
```

Method Type:

```
    public void setAnimal (String a) {
        type = a;
    }
```

Method Type:

```
    public boolean equals (Animal two) {
      if (two.getAnimal ().equals (type) &&
            two.getSound ().equals (noise))
         return true;
      else
         return false;
    }
```

Method Type:

```
    public int compareTo (Animal two) {
    if (two.equals (this))
        return 0;
     else if (two.getAnimal ().compareTo (type)
            >= 0)
        return 1;
     else
            return -1;
    }
}
```

2. Fill in the blanks to create a "person" object.

```
public class _________________ {
  //Instance variables
    private String name;
    private int age;
  //Constructor I: Assign defaults
    public _________________ () {
        name = "_________________";
        age = 18;
    }
  //Constructor II: Assign in values
    public _____________ (String n, int a) {
        ___________________ = n;
        ___________________ = a;
    }
  //Accessor: converts both variables to string
    public String toString () {
        return ____________________________________;
    }

  //Accessor: returns instance variables
    public String getName () {
        return ___________________;
    }
    public int getAge() {
        return ___________________;
    }

  //Mutator: changes instance variables
    public void setName (String n) {
        _____________ = n;
    }
    public void setAge (int a) {
        _____________ = a;
    }

  //Facilitator: If equal, return true
    public boolean equals (Person two) {
        if (two.get_________ ().equals (name) &&
           two.get______ () == age)
            return true;
        else
            return false;
    }
  //Facilitator: Compare on the basis of name
   public int compareTo (Person two) {
    if (two.equals (this))
        return 0;
    else if (two.getName ().compareTo (name) >= 0)
        return 1;
    else
        return -1;
    }
}
```

3. Using the Animal Object, fill in the object memory diagrams, then fill in the blanks of the output.

```
public void show(View view) {
    TextView textArea = (TextView)
     findViewById(R.id.TextArea);

    Animal spot = new Animal ();
    textArea.append(""+spot.toString ());
    Animal fluffy = new Animal ("dog", "bark");
    textArea.append("\n"+fluffy.toString ());
    textArea.append("\n"+fluffy.getSound ());
    fluffy.setSound ("woof");
    textArea.append("\n"+fluffy.toString ());
    textArea.append("\n"+spot.compareTo (fluffy));
    textArea.append("\n"+spot.equals (fluffy));
}
```

Output:

the cat says _____

the _____ says bark

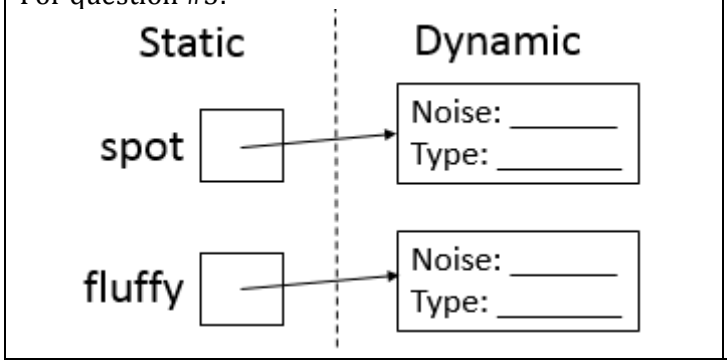
_____

the _____ _____ _____

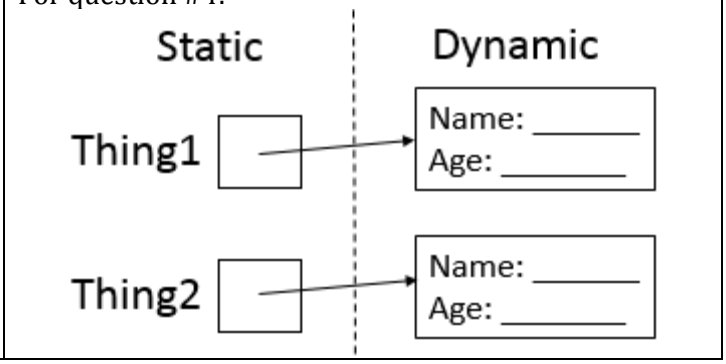
1

_____

For question #3:



For question #4:



4. Using the Person Object, fill in the object memory diagram, then fill in the blanks of the output.

```
public void show(View view) {
  TextView textArea = (TextView)
     findViewById(R.id.TextArea);

  Person Thing1 = new Person ();
  textArea.append(Thing1.toString ());
  Person Thing2 = new Person ("Bob", 88);
  textArea.append("\n"+ Thing2.toString ());
  textArea.append("\n"+ Thing2.getAge ());
  Thing2.setName ("Harpreet");
  textArea.append("\n"+ Thing2.toString ());
  textArea.append("\n"+ Thing1.compareTo (Thing2));
  textArea.append("\n"+ Thing1.equals (Thing2));
}
```

Output:

_____

_____

_____

_____

_____

_____

5. What is an object? (circle 2)
(a) data and methods that act upon that data
(b) a graphics method
(c) a (non-primitive) type
(d) a physical thing
(e) an accessor

6. What is a class? (circle 1)
(a) A set of students
(b) A biological classification system.
(c) A course taken at school
(d) A template for an object

7. Classify these methods:

a) s.**equals**(b) _____
b) textArea.append (s.**toString()**); _____
c) s.**getColor**(Color.blue); _____
d) JLabel bob = new **JLabel**("Hi"); _____
e) k.**getKleenex**(); _____
f) max.**setSize**("medium"); _____
g) if(a.**compareTo**(b) > 0) _____
h) **findViewById**(R.id.TextArea) _____
i) textArea.**setText**("hi"); _____

Objects Practice

4.4 T

Name: _____

1. Fill in the blanks to create a "Month" object.

```
public class _______________ {
```

Method Type:

```
  private String name;
  private int order;
```

Method Type:

```
  public _______________ () {
    name = "Jan";
    order = 1;
  }
```

Method Type:

```
  public _______________ (String n, int or) {
     _________________ = n;
     _________________ = or;
  }
```

Method Type:

```
  public String toString () {
     return ____________________________________;
  }
```

Method Type:

```
  public String getName () {
     return _________________;
  }
  public int getOrder() {
     return _________________;
  }
```

Method Type:

```
  public void setName (String n) {
      _____________ = n;
  }
  public void setOrder (int or) {
      _____________ = or;
  }
```

Method Type:

```
  public boolean equals (Month two) {
    if (name.equals (two._____________()) &&
          order == two.get_____________())
      return true;
    else
      return ____________;
  }
```

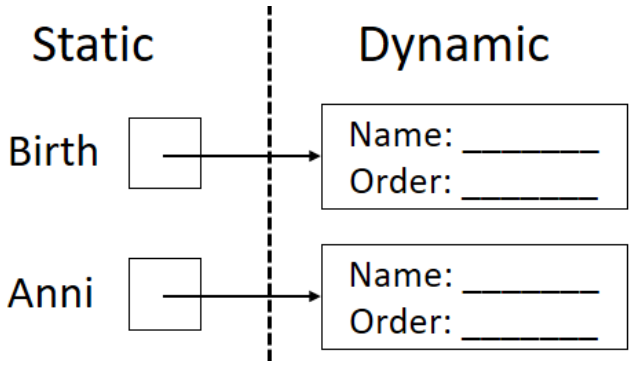
Method Type:

```
  public int compareTo (Month two) {
   if (order == two.get_____________())
      return 0;
   else if (order > two.get_________())
      return 1;
   else
      return -1;
  }
}
```

2. Using the Month Object, fill in the object memory diagrams, then fill in the blanks of the output.

```
public void show(View view) {
  TextView tA = (TextView)
     findViewById(R.id.TextArea);

  Month birth = new Month ();
  tA.append(""+birth.toString ());

  Month anni = new Month ("Feb", 2);
  tA.append("\n"+anni.toString ());
  tA.append("\n"+anni.getOrder ());

  anni.setOrder (12);

  tA.append("\n"+anni.toString ());
  tA.append("\n"+birth.equals (anni));
}
```

Object Memory Diagram:



Output:

_____

_____

_____

_____

_____

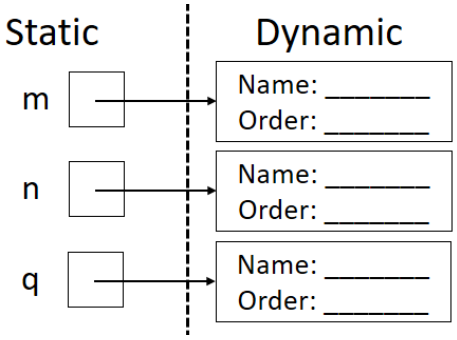
3.Consider the following String operations. Fill in the blanks and use it to circle the result.

```
String ap = "apple";                    First >  Second   => 1
String ban = "banana";                  First == Second   => 0
String cant = "cantaloupe";             First <  Second   => -1
```

Expression	First's value (look above, fill in)	Relation (circle)	Second's value (look above, fill in)	CompareTo Result (circle)
`ap.compareTo("peach");`		> = <		1, 0 -1
`ap.compareTo("aaaa");`		> = <		1, 0 -1
`ban.compareTo("plum");`		> = <		1, 0 -1
`ban.compareTo(ap);`		> = <		1, 0 -1
`ap.compareTo(ban);`		> = <		1, 0 -1
`ban.compareTo(cant);`		> = <		1, 0 -1
`cant.compareTo(ap);`		> = <		1, 0 -1
`ap.compareTo(cant);`		> = <		1, 0 -1

4. Use the Month declarations to fill in the diagram and output.

```
Month m = new Month("Apr", 4);
Month n = new Month("May", 5);
Month q = new Month("May", 5);
```



What is printed to the TextView in each case?

```
tA.append("\n"+m.compareTo(n));   _____
tA.append("\n"+n.compareTo(m));   _____
tA.append("\n"+n.compareTo(q));   _____
```

5. Write the compareTo function for the following classes.

```
public class Mark{
  int level;
  public Mark (int m){
    level = m;
  }
  public int getMark(){
    return level;
  }
```

```
public int _____ (_____ _____){
  if (_____ == _____.get_____())
    return 0;
  else if (_____ > _____.get_____())
    return 1;
  else
    return -1;
  }
```

```
public class Course {
  String name;
  public Course (String n) {
    name = n;
  }
  public String getName() {
    return name;
  }
```

6. Classify the following methods as constructor, mutators, accessors, facilitators.

(a)	setText		(e)	setBackground	
(b)	new TextView		(f)	findViewById	
(c)	Append		(g)	new Toast	
(d)	getText		(h)	setImageResource	

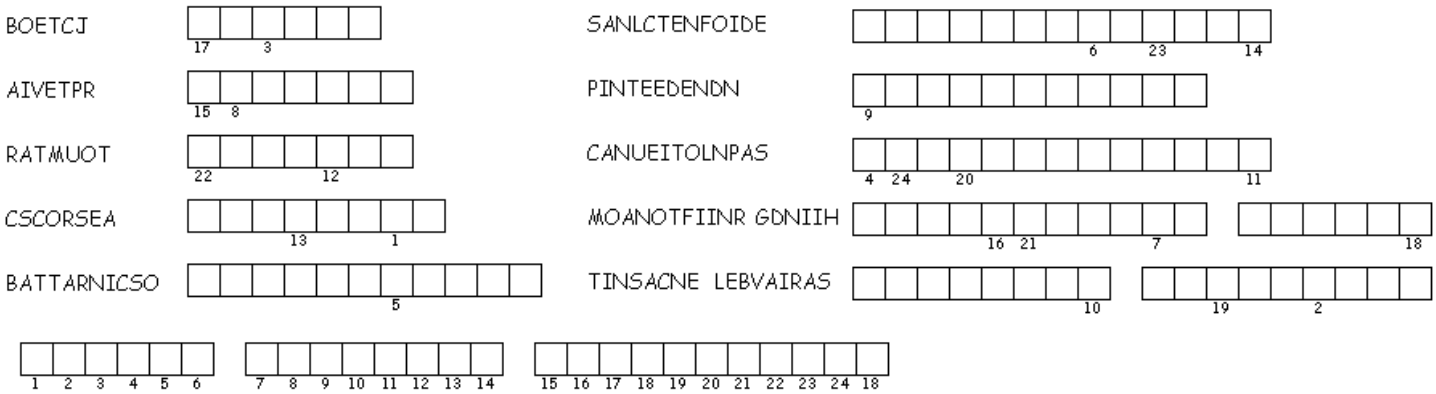
Objects Terminology 4.5 C

Name: _____

1. Fill in the appropriate term for each definition. (Information Hiding, Abstraction, Encapsulation)

	• Removing details of instance variables from user/other programmers. • Using private for instance variables. • Instance variables can only be used through accessors and mutators. • Keeps the code stable because other coders can't change instance variables.
	• Simple for other programmers to instantiate in their own programs. • Programmers can think of the problem at a higher, more removed level. • It makes using the class easy because you don't need to understand the details of how it is implemented.
	• An object's code is self-contained and independent of other code. • It relies only on itself. • Objects are easy to move around and use. It also makes them easy to update.

2. Unscramble the words from the above definitions.



3. Fill in the appropriate term for each definition. (Information Hiding, Abstraction, Encapsulation)

	a)	An object groups together instance variables and methods.
	b)	An object's code is self-contained and independent of other code.
	c)	Because an object is independent, it is easy to update.
	d)	By grouping code in logical units, they keep code organized.
	e)	Classes are easy for groups of coders to use to work together.
	f)	Classes like Toast and TextView are easy for novice coders to use them without deep understanding.
	g)	Mutators are the only route to modify instance variables.
	h)	Only accessors allow other coders to access instance variables.
	i)	Other programmers can use the code easily without understanding it.
	j)	Other programmers can't change the instance variables and make the code unstable.
	k)	Self-contained objects are easy to attach to other programs.
	l)	The details of how the class is coded are hidden from other programmers.
	m)	The use of private instance variables.

4. Fill in the blanks to create a "Card" object.

```
public class _______________ {
```

Method Type:

```
   private char suit;
   private int value;
```

Method Type:

```
  public _______________ () {
    suit = "h"; //Ace of hearts
    value = 1;
  }
```

Method Type:

```
  public _______________ (char s, int v) {
     _________________ = s;
     _________________ = v;
  }
```

Method Type:

```
  public String toString () {
     return ____________________________________;
  }
```

Method Type:

```
  public ________ getSuit () {
     ____________________________;
  }
  public ________ getValue() {
     ____________________________;
  }
```

Method Type:

```
  public _________ setSuit (char s) {
      ____________________________;
  }
  public _________ setValue (int v) {
      ____________________________;
  }
```

Method Type:

```
  public ___________ equals (Card c) {
    if (_____________________________ &&
          _____________________________)
       return true;
     else
       return ___________;
  }
```

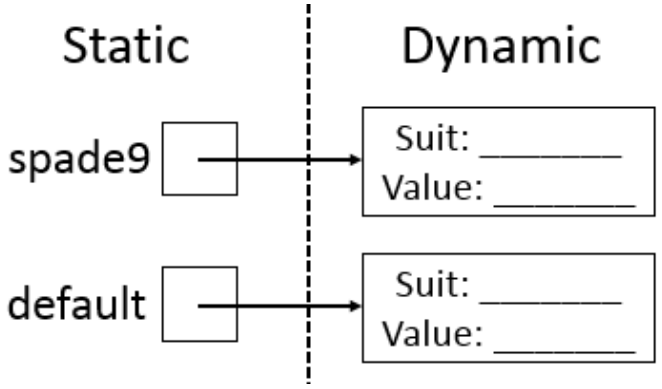
Method Type:

```
  public _________ compareTo (Card c) {
  //based on card value
   if (_____________________________)
      return 0;
   else if (_____________________________)
      return 1;
   else
      return -1;
  }
}
```

5. Using the Month Object, fill in the object memory diagrams, then fill in the blanks of the output.

```
public void show(View view) {

TextView tA = (TextView)
     findViewById(R.id.TextArea);

Card spade9 = new Card ('s', 9);
tA.append(""+spade9.toString ());

Card default = new Card ();
tA.append("\n"+default.toString ());
tA.append("\n"+spade9.getSuit ());

default.setValue (7);

tA.append("\n"+default.toString ());
tA.append("\n"+default.equals (spade9));

}
```

Object Memory Diagram:



Output:

_____

_____

_____

_____

_____

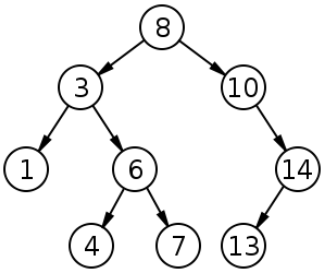
Stacks

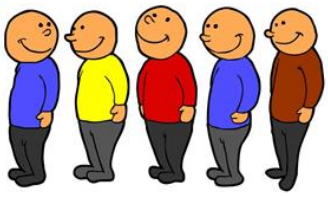
4.6 c

Name: _____

1. Circle the images that represent Stacks.








2. Circle the Stack methods.

enqueue	pop	dequeue	add	addtoTail	peek
isEmpty	isFull	push	remove	removeformTail	equals

3. Note what is printed out beside each line of output.

```
Stack s = new Stack ();
s.push("9");
s.push("8");
TextArea.append(""+s.pop());          _____
s.push("7");
TextArea.append(""+s.pop());          _____
TextArea.append(""+s.size ());        _____
s.push("6");
TextArea.append(""+s.pop ());         _____
TextArea.append(""+s.pop ());         _____
textArea.append(""+s.isEmpty ());     _____
```

```
Stack t = new Stack ();
t.push("10");
t.push("11");
TextArea.append(""+t.size ());        _____
TextArea.append(""+t.pop());          _____
t.push("12");
TextArea.append(""+t.pop());          _____
TextArea.append(""+t.size ());        _____
t.push("13");
TextArea.append(""+t.peek ());        _____
TextArea.append(""+t.pop ());         _____
```

4. Circle and correct five errors in this code.

```
Stack s1 = new Stack ();
Stack s3 = new Stack1 ();

s1.push ("12");
s1.push (25);
s1.push ("45");

s1.pop ();
s1.addtoTail ("97");
```

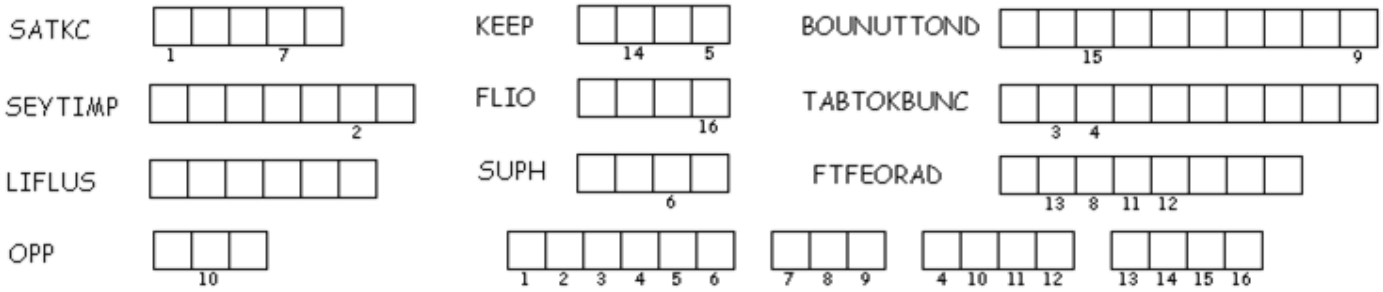
```
        int c = s1.size ();
        for (int i = 0 ; i < c ; i++) {
            String t = (String) s1.pop ();
            TextArea.append(""+t);
            s3.add (t);
        }
        while(!s3.notEmpty()) {
            s1.push (s3.pop ());
        }
      }
}
```

5, Find the words in this word search.

O Q F Q X U F M S P C W P Y
W K U P N D W F X T O L U T
U N D O B U T T O N A P S P
I X H B L C L J E E U C H M
W C L Z R V C M U U D P K E
N O T T U B K C A B E A N S
Z L L T Q Q W R O E D W R I
I S F U L L R Z K O F I L T

- ❑ ISEMPTY
- ❑ ISFULL
- ❑ PEEK
- ❑ POP
- ❑ PUSH
- ❑ STACK
- ❑ TRADE OFF
- ❑ UNDO BUTTON
- ❑ BACK BUTTON
- ❑ LIFO

6. Unscramble the following words, they may be starting to seem very familiar….that is no accident!



7. What is the speed of each Stack operation (in Big-Oh notation)?

push:		pop:		peek:	
clear:		isEmpty:		size:	

8. Stack operations are very restricted. Circle the 3 things that can be modelled using a stack.
   a) The undo button. It pops off the last task which occurred and resets it.
   b) A line of people waiting at the grocery store.
   c) A Sudoku board.
   d) FIFO – first in, first out situation.
   e) LIFO – last in, first out situation.
   f) Recursive calls. Each call is added to a run-time stack.
   g) Sorting the list into alphabetical order.
   h) Searching for an item and finding its position.

9. What are the benefits of restricting the stack operations? (Circle 1 answer)
   a) It keeps the object code shorter and easier to understand.
   b) By only having a limited number of options, stack operations can be optimized to all be constant time.
   c) Limiting the options makes the code more versatile.
   d) It is more visually appealing when it has fewer options.

10. Why is sorting a Stack a bad idea?

. . . . . .

. . . . . .

. . . . . .

. . . . . .

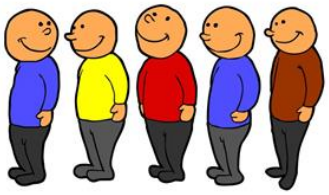
Queue

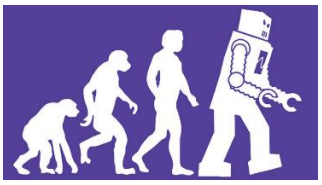
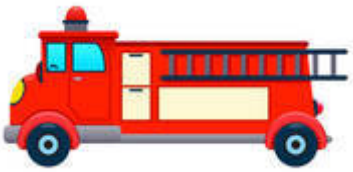
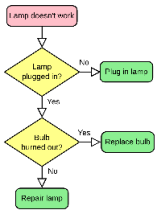
4.7 C

Name: _____

1. Circle the images that represent queues.





2. Circle the queue methods.

enqueue	pop	dequeue	add	addtoTail	peek
isEmpty	isFull	push	remove	removefromTail	equals

3. Skunk Bingo has a log where you enter the pieces in and order and they exit the other end in the same order.

(a) The Log in Skunk Bingo starts with these pieces.
Fill in the code.

Back Front

```
Queue q = new Queue();
q.enqueue("bird");
q.enqueue("_____");
q.enqueue("_____");
q.enqueue("_____");
q.enqueue("_____");
q.enqueue("_____");
```

(b) Four cards are removed and four new cards are added.
Fill in the code.

Back Front

```
q.dequeue();
q.dequeue();
q.dequeue();
q.dequeue();
q.enqueue("_____");
q.enqueue("_____");
q.enqueue("_____");
q.enqueue("_____");
```

(c) Four cards are removed and four new cards are added.
Fill in the code.

Back Front

```
q._____();
q._____();
q._____();
q._____();
q.enqueue("_____");
q.enqueue("_____");
q.enqueue("_____");
q.enqueue("_____");
```

4. Note what is printed out beside each line of output.

```
Queue q = new Queue ();
q.enqueue (9);
q.enqueue (8);
System.out.println (q.dequeue ());  ____
q.enqueue (7);
System.out.println (q.size ());     ____
System.out.println (q.dequeue ());  ____
System.out.println (q.dequeue ());  ____
System.out.println (q.isEmpty ());  ____
```

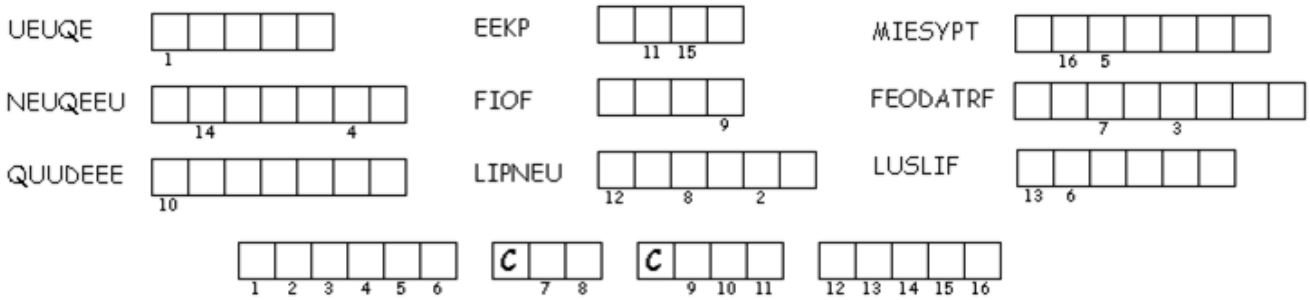
```
Queue q = new Queue ();
q.enqueue (10);
System.out.println (q.dequeue ()); ____
System.out.println (q.size ());    ____
q.enqueue (12);
System.out.println (q.isEmpty ()); ____
System.out.println (q.dequeue ()); ____
q.enqueue (14);
System.out.println (q.dequeue ()); ____
```

5. Search for the queue related words.

I	D	K	E	B	B	I	D	H	O
S	W	E	F	U	S	B	O	F	F
E	J	E	Q	F	E	D	J	F	G
M	G	P	U	U	W	U	O	A	W
P	T	L	O	L	E	E	Q	J	S
T	L	F	Y	Z	D	U	L	Q	Y
Y	I	Y	D	A	B	I	E	X	B
F	E	M	R	P	U	E	N	I	L
K	U	T	E	N	Q	U	E	U	E
K	Q	J	B	P	L	G	P	Q	V

- ❑ DEQUEUE
- ❑ ENQUEUE
- ❑ FIFO
- ❑ ISEMPTY
- ❑ ISFULL
- ❑ LINEUP
- ❑ PEEK
- ❑ QUEUE
- ❑ TRADEOFF

6. Unscramble the queue functions and use the words to fill in the puzzle.



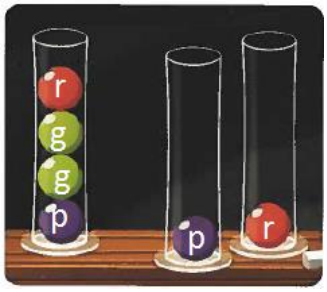
7. Fill in the queue words that match the clues given.

	(a) The name of a data type that holds elements in the same order as a line.
	(b) First In, First Out.
	(c) Adding something to a queue.
	(d) Removing something from a queue.
	(e) Checking if the queue has no space left.
	(f) Checking if the queue has nothing in it.
	(g) All of the functions are very quick, but they are limited to FIFO operations.
	(h) Queues can be used to represent this real world operation.
	(i) Peeking at the top item in the queue.

More About Stacks and Queues 4.8 T

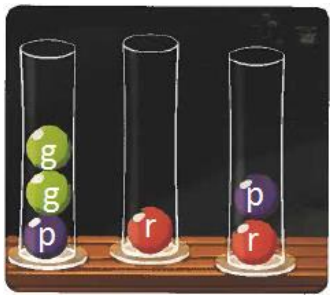
Name: _____

1. (a) Fill in the code to make these 3 Stacks.



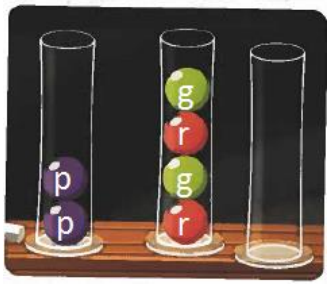
```
s1.push("____");
s1.push("____");
s1.push("____");
s1.push("____");

s2.push("____");

s3.push("____");
```

(b) Fill in the code. Change the previous to this picture.



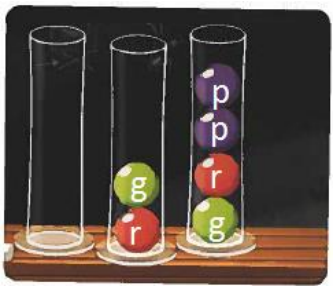
```
s3.push(s2.pop());

___.push(s1.pop());
```

2. (a) Fill in the code to make these 3 Stacks.



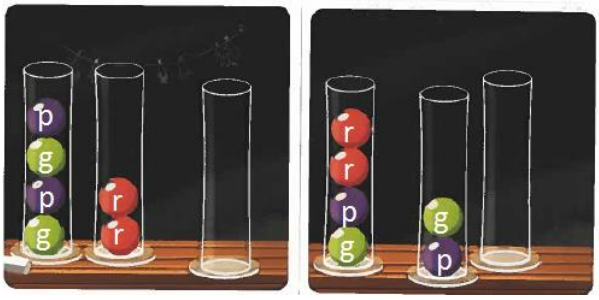
```
s1.push("____");
s1.push("____");

s2.push("____");
s2.push("____");
s2.push("____");
s2.push("____");
```

(b) Fill in the code. Change the previous to this picture.



```
s3.push(___.pop());
s3.push(___.pop());

s3.push(___.pop());
s3.push(___.pop());
```

What is the smallest number of steps you can take to transfer the stack in the first picture to the second?



Smallest number of steps?. . . . . .

4. Use Stack terminology to fill in the blanks.

	(a) A structure that stores data in a LIFO order. Only the top can be accessed.
	(b) Checks if a stack has no more room left.
	(c) Checks if a stack has nothing in it.
	(d) Peeks at the top item of the stack.
	(e) Takes an item off the Stack.
	(f) Puts an item on the Stack.
	(g) Last in, first out.
	(h) An application of a Stack inside MS Word and many other applications.
	(i) A good thing is that all stack operations are very fast. However, the reason that the stack operations are fast is that they are limited to LIFO operations.

5. Choose the best data structure for each description (Stack or Queue or Both).

_____ a. A data structure where you add to the back and remove from the front.

_____ b. A data structure where you add to the back and remove from the back.

_____ c. A data structure with the method Peek()

_____ d. A pile of books model this data structure well.

_____ e. A LIFO structure.

_____ f. The data structure that would hold print jobs for a printer.

_____ g. The data structure used by the undo button in Excel.

_____ h. A FIFO structure.

6. Over time, a Stack has 1, 2, 3 pushed on it.
What orders can you make the numbers pop? If it is possible, code it. If not, note that it is not possible.

1-2-3
`s.push(1)`
`s.pop()`
`s.push(2)`
`s.pop()`
`s.push(3)`
`s.pop()`
	2-3-1	**3-2-1**

1-3-2	**2-1-3**	**3-1-2**

7. Over time 1,2,3 are enqueued onto a queue.
(a) Can they be dequeued in the 6 possible orders? Which ones are possible? (circle)

1-2-3	2-3-1	3-2-1	1-3-2	2-1-3	3-1-2

(b) WHY aren't the others possible?

. . . . . .

. . . . . .

. . . . . .

Objects & ADTs, Part 1

4.9 K

Name: _____

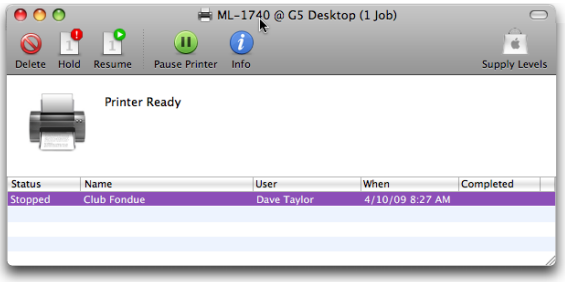
1. These three classes simulate a series of addition questions on a stack of flash cards.

   Fill in the MathQuestion – the class to hold each flash card. Adapt the Stack class to hold a Stack of MathQuestions. Fill in the Runner class to add three questions to the stack.

```
public class MathQuestion {
	private int n1;
	private int n2;

	public MathQuestion(){
		n1 = (int)(Math.random()*5);
		n2 = (int)(Math.random()*5);
	}
	public MathQuestion(int range){
		n1 = (int)(Math.random()*range);
		n2 = (int)(Math.random()*range);
	}
	public String getQuestion(){
	//returns question in format 3 + 4 = ?









	}
	public int getAnswer(){
		return (n1+n2);
	}
	public void newQuestion(){
		n1 = (int)(Math.random()*5);
		n2 = (int)(Math.random()*5);
	}
	public void newQuestion(int range){









	}
	public boolean checkAnswer (int guess){
	//return true if guess equals the answer
	//false otherwise









	}
}
```

```
public class Stack {

private int count;

private Object data[] = new Object [50];

    public Stack () {
        count = 0;
    }

    public void push (Object addMe)  {
        data [count] = addMe;
        count++;
    }

    public int size ()  {
        return count;
    }

    public boolean isFull ()  {
        return (count == 50);
    }

    public Object pop () {{
        count--;
        return data [count];
    }

    public Object peek () {
        return data [count--];
    }

    public boolean isEmpty () {
        return count == 0;
    }

    public void clear () {
        count = 0;
    }
}
```

```
Stack s = new Stack();
MathQuestion q = new MathQuestion();
s.push(q);
//Add two more questions
```



2. This code simulates a printer queue.
   (a) Fill in the PrinterJob Object
   (b) Adapt the Queue class so that it stores a queue of PrintJobs.
   (c) Create a Queue. Add two PrintJobs to the Queue.

```
public class PrinterJob {

    private String filename;
    private double time;

    public PrinterJob(){



    }
    public PrinterJob(String fn, double t){



    }
    public void setFileName(String fn){

    }
    public void setTime(double t){

    }
    public String getFileName(){

    }
    public double getTime(){

    }
    public boolean equals(PrinterJob j){









    }
    public int compareTo(PrinterJob j){
        if(time < _________________)
            return -1;
        else if(time == _________________)
            return 0;
        else
            return 1;
    }
    public String toString(){
    //returns a phrase like Temp1.doc was
    //  added to the printer queue at 3.30



    }
}
```

```
public class Queue {

  private Object data[] = new Object [50];
  private int count;
  private int head;

  public Queue () {
    count = 0;
    head = 0;
  }

  public void enqueue (Object value) {
    int tail = (head + count) % data.length;
    data [tail] = value;
    count++;
  }

  public Object dequeue () {
    Object temp = data [head];
    count--;
    head = (head + 1) % data.length;
    return temp;
  }

  public Object peek () {
    return data [head];
  }

  public int size () {
    return count;
  }

  public boolean isEmpty () {
    return (count == 0);
  }

}
```

```
Create a Queue.
Add two Print Jobs to the queue.
```

Objects & ADTs, Part 4.10 K

Name: _____

3. "Stack" is a simple dice rolling and collecting game. The die are rolled. Each player gets one colour of dice. Players stack dice that have rolled the same number on top of their opponents dice to "capture" stacks. Once a stack of dice reaches 4 high, it is frozen and points awarded to the owning player. Players have the option of re-rolling a die before stacking it.

(a) Fill in the Dice Class

```
public class Dice {
	private int top;

	public Dice(){
		top = (int)(Math.random()*6+1);
	}
	public void rollDice(){



	}
	public int getTop(){



	}
	public boolean equals(Dice d){









	}
	public int compareTo(Dice d){

















	}
	public String toString(){
		return "The dice rolled a "+top;
	}
}
```

(b) Adapt the Stack to make a Stack of Dice

```
public class Stack {

private int count;

private Object data[] = new Object [50];

    public Stack () {
        count = 0;
    }

    public void push (Object addMe)  {
        data [count] = addMe;
        count++;
    }

    public int size ()  {
        return count;
    }

    public boolean isFull ()  {
        return (count == 50);
    }

    public Object pop () {{
        count--;
        return data [count];
    }

    public Object peek () {
        return data [count--];
    }

    public boolean isEmpty () {
        return count == 0;
    }

    public void clear () {
        count = 0;
    }
}
```

(c) Create a Stack of Dice. Add 4 Dice to the stack.

4. Queue is a game that teaches students about food rationing systems.

(a) Create a player class. It stores a colour and a name.

(b) Adapt the queue class to store a queue of players (as shown in the picture)

(c) Create a queue. Add 3 different player tokens to the queue.


```
public class Player {
       private String colour;
       private String name;

       public Player(){


       }
       public Player(String c, String n){


       }
       public String getName(){

       }
       public String getColour(){

       }
       public void setName(String n){

       }
       public void setColour(String c){

       }
       public boolean equals(Player p){









       }
       public int compareTo(Player p){









       }
       public String toString(){
       //returns a phrase like:
       //   the red piece is next (Gorski)




       }
}
```

```
public class Queue {

  private Object data[] = new Object [50];
  private int count;
  private int head;

  public Queue () {
    count = 0;
    head = 0;
  }

  public void enqueue (Object value) {
    int tail = (head + count) % data.length;
    data [tail] = value;
    count++;
  }

  public Object dequeue () {
    Object temp = data [head];
    count--;
    head = (head + 1) % data.length;
    return temp;
  }

  public Object peek () {
    return data [head];
  }

  public int size () {
    return count;
  }

  public boolean isEmpty () {
    return (count == 0);
  }

  public void clear () {
    count = 0;
  }
}
```

```
Create a queue.
     Add three different players to it.
```

UMLs and Memory Diagrams 4.11

Name: _____

1. Adapt the Queue so that it stores Customers.

```
public class Queue {

  private Object data[] = new Object [4];
  private int count;
  private int head;

  public Queue () {
    count = 0;
    head = 0;
  }

  public void enqueue (Object value) {
    int tail = (head + count) %
      data.length;
    data [tail] = value;
    count++;
  }

  public Object dequeue () {
    Object temp = data [head];
    count--;
    head = (head + 1) % data.length;
    return temp;
  }

  public Object peek () {
    return data [head];
  }

  public int size () {
    return count;
  }

  public boolean isEmpty () {
    return (count == 0);
  }
}
```

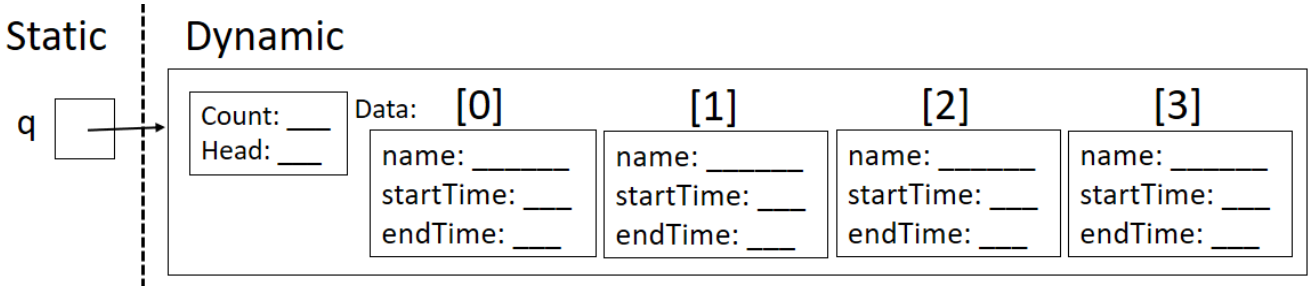
In MainActivity.java :

```
Queue q = new Queue();
Customer c = new Customer();
q.enqueue(c);
q.enqueue(new Customer("Gupta", 10, 11);
q.enqueue(new Customer("Zhao", 11, 13);
```

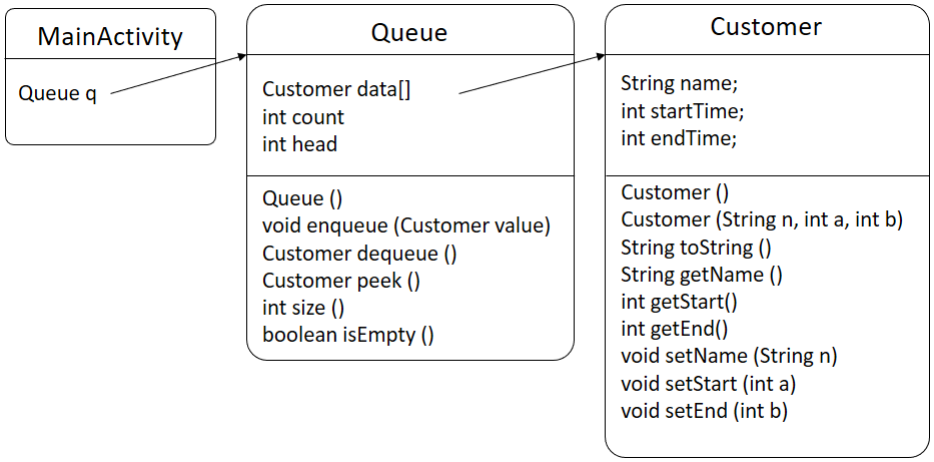
2. Fill in the Customer class.

```
public class Customer {
  //Instance variables
    private String name;
    private int startTime;
    private int endTime;

  //Constructor I: Assign defaults
    public _______________ () {
        name = "Gorski";
        startTime = 9;
        endTime = 10;
    }
  //Constructor II: Assign in values
  public _______________ (String n, int a, int b) {
        ________________ = n;
        ________________ = a;
        ________________ = b;
    }
  //Converts all variables to string
    public String toString () {
        return ____________________________________
          ____________________________________;
    }
  //Returns instance variables
    public String getName () {
        return _________________;
    }
    public int getStart() {
        return _________________;
    }
    public int getEnd() {
        return _________________;
    }
  //Changes instance variables
    public void setName (String n) {
        _____________ = n;
    }
    public void setStart (int a) {
        _____________ = a;
    }
    public void setEnd (int b) {
        _____________ = b;
    }
}
```

3. Using the MainActivity Class, fill in the memory diagram for the Queue of Customers.

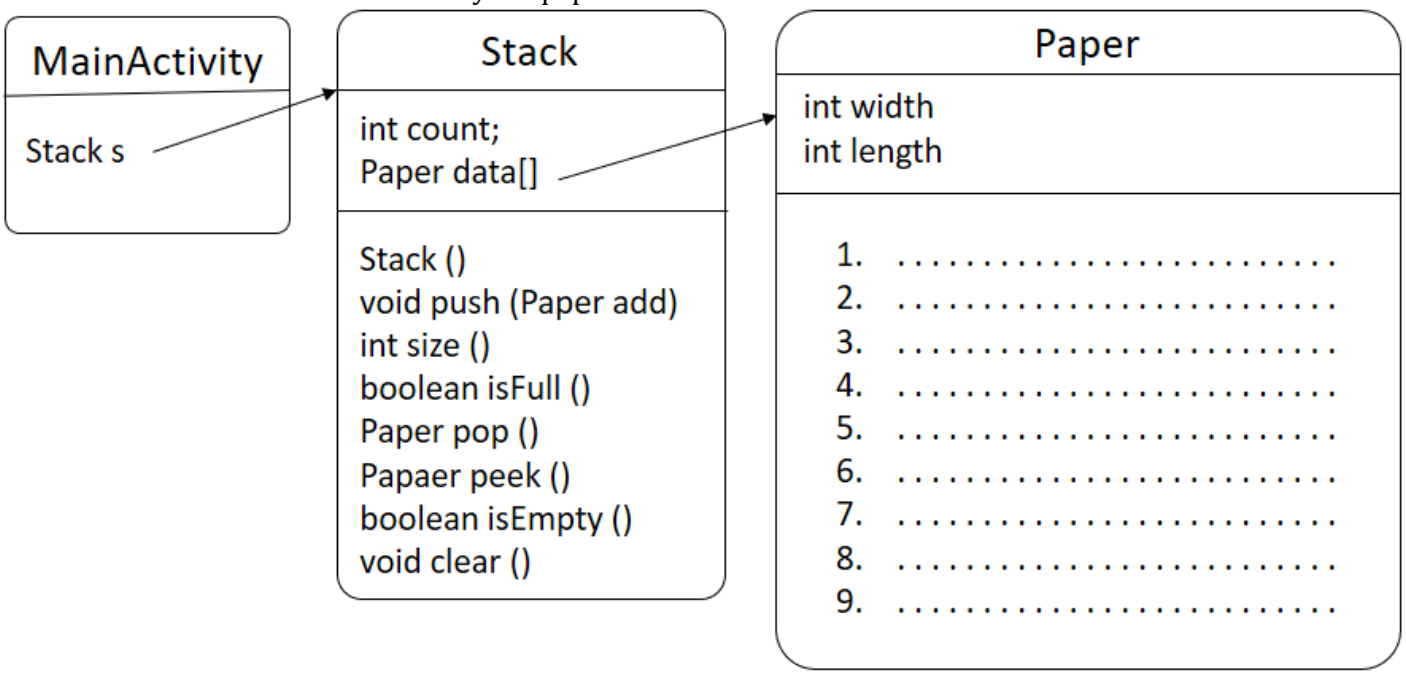


4. This is a UML diagram for the classes on the front page.



a) How many classes?	
b) How many instance variables in Queue?	
c) How many things are private in Customer?	
d) How many things are public in Queue?	
e) How many methods are in Customer?	
f) How many constructors in Customer?	
g) How many constructors in Queue?	
h) How many accessors in Customer?	

5. What 9 methods would be needed by the paper class? Fill them in the UML.



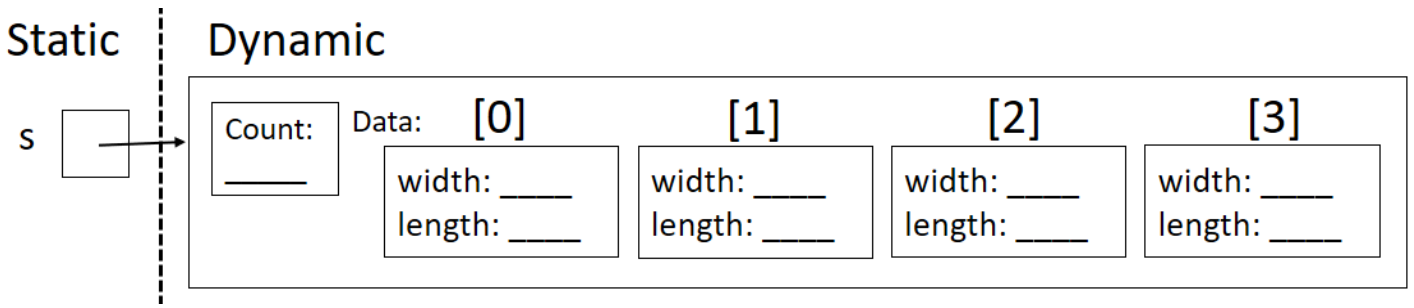
6. Fill in memory diagram for the MainActivity that creates a Stack of Paper.

In the MainActivity:

```
public void run(View view) {
  Stack s = new Stack();
  s.push(new Paper(3, 4);
  s.push(new Paper(5, 6);
  s.pop();
  s.push(new Paper(7, 8);
  s.push(new Paper (9, 10); }
```

A selection of the Stack Code:

```
private int count;
private Paper data[] = new Paper [4];
public Stack () { count = 0; }
public void push (Paper add){
   data [count] = add;  count++; }
public Paper pop () {
   count--; return data [count]; }
```

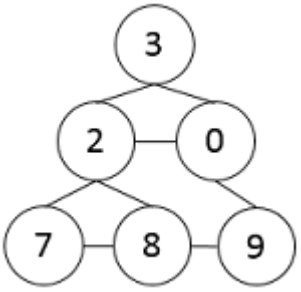


Binary Search Trees

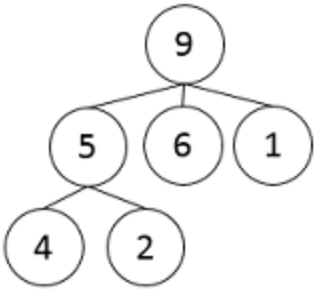
4.12 C

Name: _____

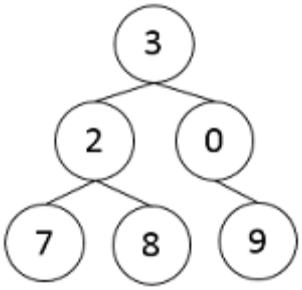
1. For each graph, check off all of the terms that apply.



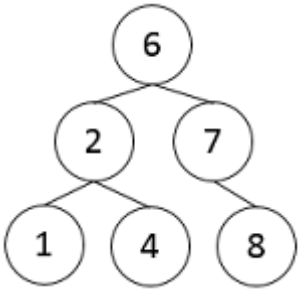
- ❑ Graph
- ❑ Tree
- ❑ Binary Tree
- ❑ Binary Search Tree



- ❑ Graph
- ❑ Tree
- ❑ Binary Tree
- ❑ Binary Search Tree



- ❑ Graph
- ❑ Tree
- ❑ Binary Tree
- ❑ Binary Search Tree



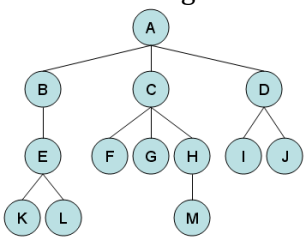
- ❑ Graph
- ❑ Tree
- ❑ Binary Tree
- ❑ Binary Search Tree

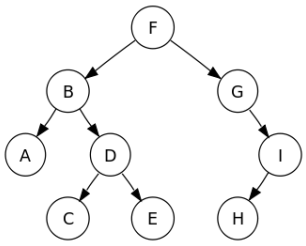
2. Match the following terms with the definitions.

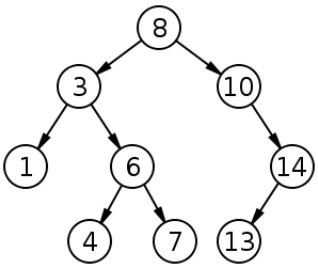
Tree, Binary Tree, Binary Search Tree, Root, Leaf, Vertex, Edge

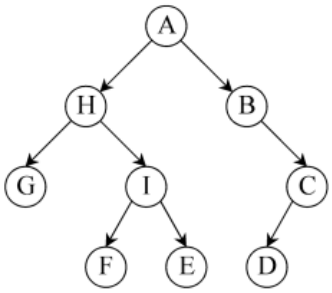
	(a) A structure where all vertices are connected. There are no simple circuits.
	(b) A dot or node that is connected to others.
	(c) A vertex that has no descendants.
	(d) The vertex with no parents.
	(e) Lines that connect vertices.
	(f) A tree where each vertex has a maximum of two children.
	(g) A binary tree where all the vertices are smaller than those found to the right of them.

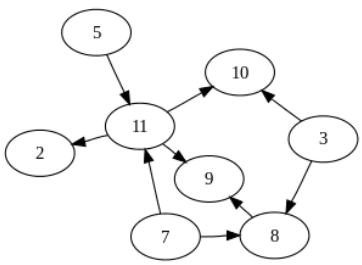
3. Circle the images that represent binary search trees.

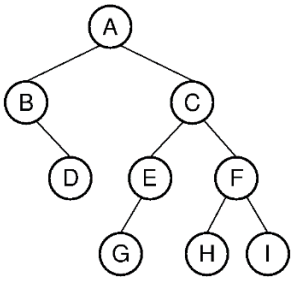


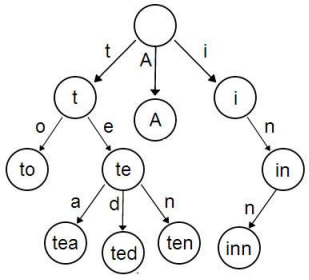




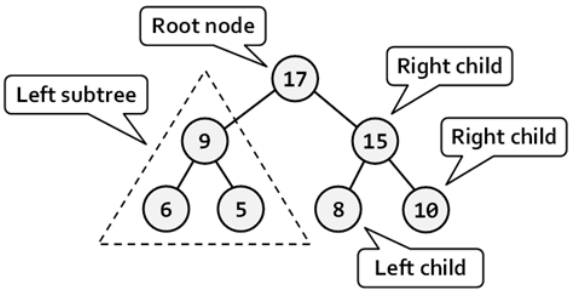


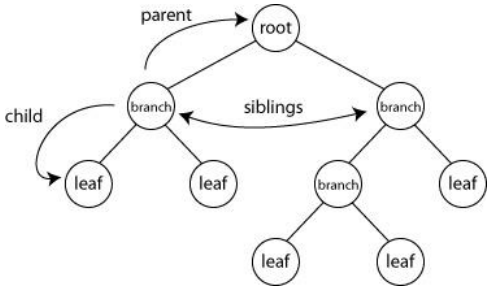






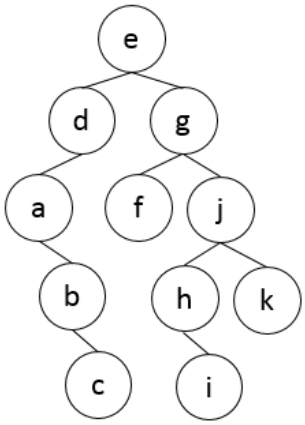
4. Look at the following binary tree terminology:



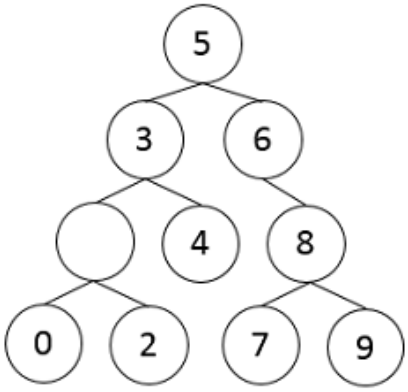


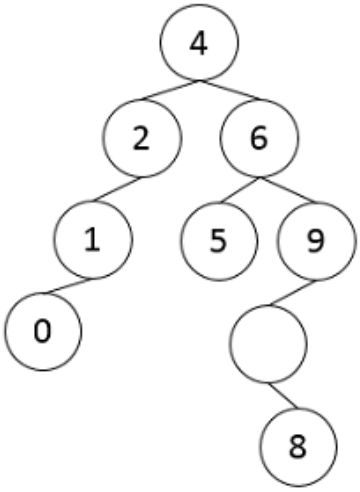
Apply the terminology to the search tree on the right.

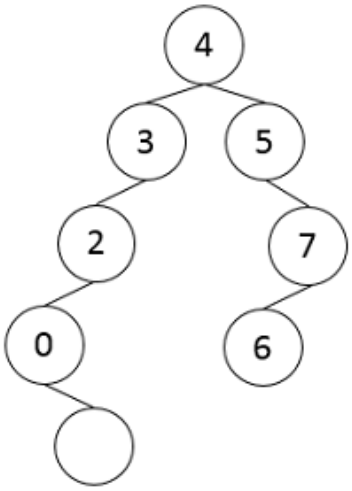
a) How many nodes? .....
b) How many leaves? .....
c) How many edges? .....
d) What is the root node? .....
e) How many vertices? .....
f) What is the child of a? .....
g) What is the right child of g? .....
h) What is h's sibling? .....
i) What is f's parent? .....
j) How many nodes in the left subtree? .....
k) How many nodes in the right subtree? .....



5. What number goes in the missing space in each binary search tree?







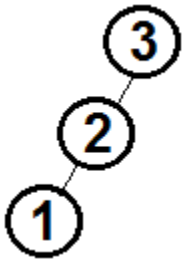
6. Create binary search trees for the following sets of data:

A, B, C, D, E	5, 4, 7, 2, 8, 9, 3, 6	cat, dog, egg, apple, kite, zebra

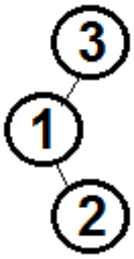
Binary Search Tree Algorithms 4.13 T

Name: _____

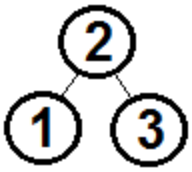
1. Overtime, 1, 2 and 3 are added in different orders to a binary search tree. Differing inserts of these three numbers can produce the following trees. Write the insert order to produce each.




```
t.insert(____);
t.insert(____);
t.insert(____);
```



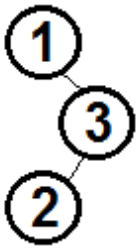

```
t.insert(____);
t.insert(____);
t.insert(____);
```



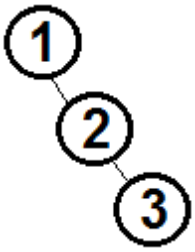

```
t.insert(____);
t.insert(____);
t.insert(____);
```

or

```
t.insert(____);
t.insert(____);
t.insert(____);
```




```
t.insert(____);
t.insert(____);
t.insert(____);
```

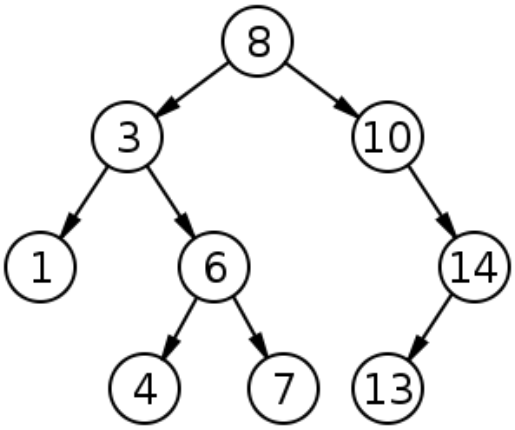



```
t.insert(____);
t.insert(____);
t.insert(____);
```

2. Consider this binary search tree:

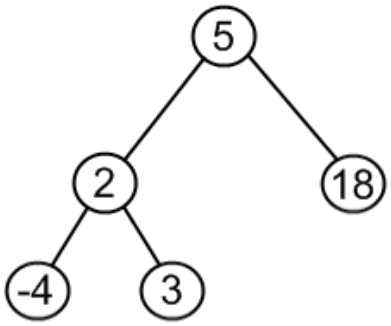
a) How many nodes? .....
b) What is the root node? .....
c) How many leaves? .....
d) What is the child of 14? .....
e) How many nodes in the left subtree? .....
f) What is 6's sibling? .....
g) What is 1's parent? .....

h) How many vertices? .....
i) How many edges? .....
j) How many parent vertices? .....

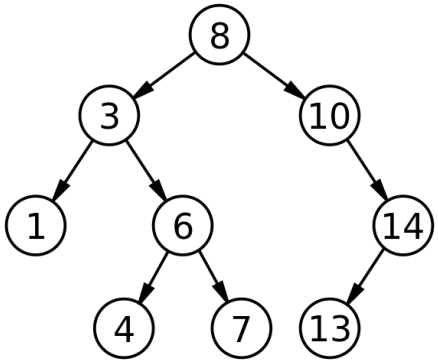


3. Highlight the nodes visited to find the value the nodes.

```
System.out.println(t.search("3"));
```



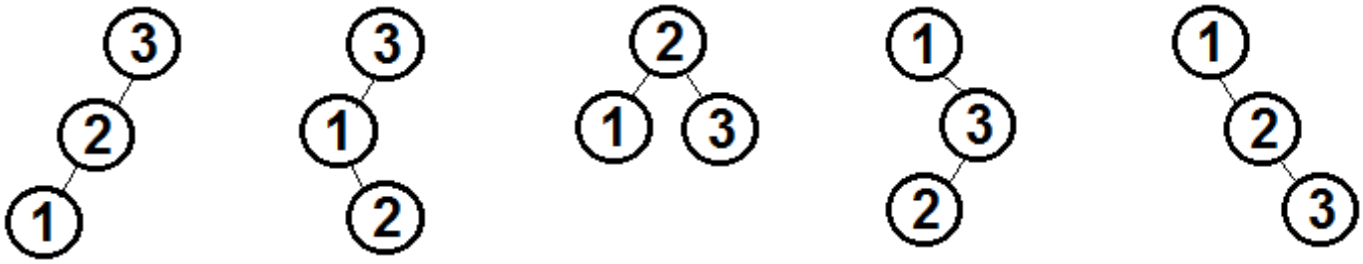

```
System.out.println(t.search("4"));
```



4. Circle true or false for each of the following:

T F a) A tree requires a root.
T F b) A binary tree must have more leaves than parent nodes.
T F c) In a binary tree, the root always has the greatest height.
T F d) When searching, it is possible to have many different paths between the root and a leaf.
T F e) The insert order determines the shape of the binary tree.
T F f) Binary trees are naturally based on while loops and iteration structures.

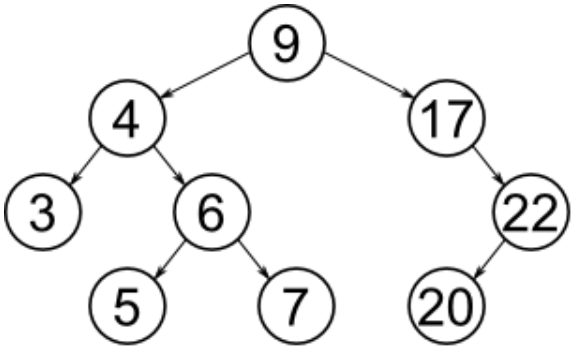
5. Consider the position of the maximum (highlight in pink) and the minimum (highlight in yellow).



6. This is the code used to find the maximum value in a binary search tree.

```
private String maxRec (Node root) {

   if (root.right != null)

     return maxRec (root.right);

   else

     return root.key;
}
```

(a) ❑ Label the base case
(b) ❑ Label the recursive case
(c) Trace what the code is doing.
Highlight the visited nodes.



7. Modify the maximum code to find the minimum.

```
    private String maxRec (Node root) {

        if (root.right != null)

            return maxRec (root.right);

        else

            return root.key;
    }
```

8. This is the code to insert a node into a BST.
(a) ❑ Label the base case
(b) ❑ Label the recursive case

```
    private Node insertRec (Node root, String key) {
        if (root == null) {
            root = new Node (key);
            return root;
        }
        else if (key.compareTo (root.key) < 0)
            root.left = insertRec (root.left, key);
        else if (key.compareTo (root.key) > 0)
            root.right = insertRec (root.right, key);
        return root;
    }
```

9. Which insert order would produce the binary search tree on the right? (circle 1)
(a) BCFGJKM
(b) MKJGFCB
(c) GKCFJMB
(d) GBCFKJM

❑ Fill in the correct values in the nodes on the diagram to the right.