1. (10 Points) What is the output of the following code?

```
public class Q1_1 {
      public static void main(String[] args) {
            foo(false, false);
            foo(false, true);
            foo(true, false);
            foo(true, true);
      }
      public static void foo(boolean a, boolean b) {
            if (a) {
                  System.out.println("A");
            } else if (a && b) {
                  System.out.println("A && B");
            } else {
                  if (!b) {
                        System.out.println("notB");
                  } else {
                        System.out.println("ELSE");
                  } // end if
            } // end if
      }
}
```

2. (10 Points) What is the output of the following code?

```
public class Q2_1 {
    public boolean [] b = new boolean[3];
    public int count = 0;

    public static void main(String [] args) {
        Q2_1 ba = new Q2_1();
        ba.set(ba.b, 0);
        ba.set(ba.b, 2);
        ba.test();
    }
    void set(boolean [] x, int i) {
        x[i] = true;
        ++count;
    }
    void test() {
        if ( b[0] && b[1] | b[2] )
            count++;
        if ( b[1] && b[(++count - 2)] )
            count += 7;
        System.out.println("count = " + count);
    }
}
```

3. (20 Points) What is the output of the following code?

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

   for (int i = 0; i < strings.length; i++) {
      strings[i] = null;
   } // end for

   strings[2] = "Lucas";  strings[5] = "Binxin"; strings[7] = "Ariel";
   strings[9] = "Lisa";   strings[4] = "Samuel"; strings[6] = "Kalambay";
   strings[0] = "Robert"; strings[1] = "Randy";  strings[8] = "Carlos";
   strings[3] = "Julio";

   int j = 3;
   for (int i = 0; i < strings.length; i++) {
      j = (i % 2) + j;
      try {
         switch (i) {
            case 0: System.out.println("j = " + j + " " + strings[i + j].toLowerCase());
            case 1: System.out.println("j = " + j + " " + strings[i - j].toUpperCase()); break;
            case 2: System.out.println("j = " + j + " " + strings[i + j].toLowerCase());
            case 3: System.out.println("j = " + j + " " + strings[i - j].toUpperCase()); break;
            case 4: System.out.println("j = " + j + " " + strings[i + j].toLowerCase());
            case 5: System.out.println("j = " + j + " " + strings[i - j].toUpperCase()); break;
            case 6: System.out.println("j = " + j + " " + strings[i + j].toLowerCase());
            case 7: System.out.println("j = " + j + " " + strings[i - j].toUpperCase()); break;
            case 8: System.out.println("j = " + j + " " + strings[i + j].toLowerCase());
            default:
               System.out.println(strings[i - 7].toUpperCase());
            } // end switch
      } catch (Exception e) {
         System.out.println("Something Wrong!!!");
      } // end try
   } // end for
}
```

4. (20 Points) When the following program is run, it produces the given output. Write the rotate function. You may use Eclipse if you wish.

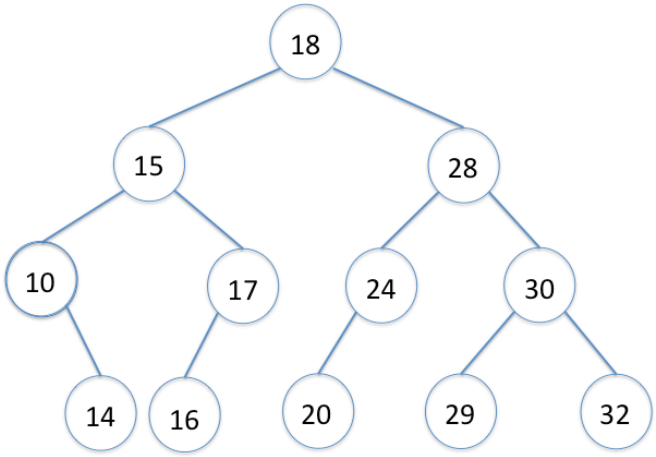
```
public class Q4 {

   public static String rotate(String s, int r) {    }

   public static void main(String[] args) {
         String name = "Sameh";
         for ( int i = 0 ; i <= name.length() ; i++ ) {
               System.out.println("When i = " + i + " - " + rotate(name,i));
         } // end for
   }
}
```

Output
When i = 0 - Sameh
When i = 1 - hSame
When i = 2 - ehSam
When i = 3 - mehSa
When i = 4 - amehS
When i = 5 - Sameh

5. (10 Points) Given that the following numbers are inserted, in the given order, into a binary tree. Draw the resulting tree.

(19, 25, 15, 22, 13, 16, 29, 11, 14, 17, 35, 21, 24, 27, 28, 26, 36, 22, 18, 15)

6. (20 Points) Given the following binary tree containing **`int`** s:



   a. (10 Points) What is the output of the following recursive method if it is initially called with the root node as a parameter:

```
public void preOrder(TreeNode n) {
   if (n != null) {
      System.out.print(n.getInt() + "  ");
      preOrder(n.getLeftChild());
      preOrder(n.getRightChild());
   } // end if
} // end preOrder
```

   b. (10 Points) What is the output of the following recursive method if it is initially called with the root node as a parameter:

```
public void postOrder(TreeNode n) {
   if (n != null) {
      postOrder(n.getLeftChild());
      postOrder(n.getRightChild());
      System.out.print(n.getInt() + "  ");
   } // end if
} // end postOrder
```

7. (10 Points) Use Eclipse to write the class **`QueueEntry`**. This class will have the following private attributes:

- *`previous`* – A pointer to the previous **`QueueEntry`**.
- *`object`* – A pointer to an **`Object`**.
- *`next`* – A pointer to the next **`QueueEntry`**.

Your **`QueueEntry`** class should have the following public methods:

- QueueEntry() - A constructor that initializes all attributes to null.
- QueueEntry(Object *object*) - A constructor that sets the object attribute to the specified *object* and the other attributes to null.
- get & set methods for all the attributes.

8. (20 Points) Use Eclipse to write the class **`Queue`**. This class will have the following private attributes:

   - *`head`* - A pointer to the first **`QueueEntry`** in the queue.
   - *`tail`* - A pointer to the last **`QueueEntry`** in the queue.

   Your **`Queue`** class should have the following public methods:

   - Queue() - A constructor that initializes the Queue to contain no entries.
   - Queue(QueueEntry *`entry`*) - A constructor that initializes the Queue to contain the specified *`entry`*.
   - Queue(QueueEntry[] *`entries`*) - A constructor that initializes the Queue to contain all the specified *`entries`*.
   - addEntry(QueueEntry *`entry`*) - A method that adds the specified *`entry`* to the back of the queue.
   - addEntries(QueueEntry[] *`entries`*) - A method that adds the specified *`entries`* to the queue in the same order as they appear in the array.
   - getFirstEntry() - A method that returns the first QueueEntry from the Queue. This method will remove the entry from the queue as well.
   - getHead() - Returns the head of the queue.
   - getTail() - Returns the tailg of the queue.
   - isEmpty() - A method that returns true if the Queue is empty, false otherwise.

1. (10 Points) What is the output of the following code?

```
public class Q1_2 {
	public static void main(String[] args) {
		foo(false, false);
		foo(false, true);
		foo(true, false);
		foo(true, true);
	}
	public static void foo(boolean a, boolean b) {
		if (b) {
			System.out.println("B");
		} else if (a && b) {
			System.out.println("A && B");
		} else {
			if (!a) {
				System.out.println("notA");
			} else {
				System.out.println("ELSE");
			} // end if
		} // end if
	}
}
```

2. (10 Points) What is the output of the following code?

```
public class Q2_2 {
    public boolean [] b = new boolean[3];
    public int count = 0;

    public static void main(String [] args) {
        Q2_2 ba = new Q2_2();
        ba.set(ba.b, 0);
        ba.set(ba.b, 2);
        ba.test();
    }
    void set(boolean [] x, int i) {
        x[i] = true;
        ++count;
    }
    void test() {
        if ( b[0] && b[1] | b[1] )
            count++;
        if ( b[1] | b[(count - 2)] )
            count += 7;
        System.out.println("count = " + count);
    }
}
```

3. (20 Points) What is the output of the following code?

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

   for (int i = 0; i < strings.length; i++) {
      strings[i] = null;
   } // end for

   strings[2] = "Lucas";  strings[5] = "Binxin"; strings[7] = "Ariel";
   strings[9] = "Lisa";   strings[4] = "Samuel"; strings[6] = "Kalambay";
   strings[0] = "Robert"; strings[1] = "Randy";  strings[8] = "Carlos";
   strings[3] = "Julio";

   int j = 2;
   for (int i = 0; i < strings.length; i++) {
      j = (i % 2) + j;
      try {
         switch (i) {
            case 0: System.out.println("j = " + j + " " + strings[i - j].toLowerCase()); break;
            case 1: System.out.println("j = " + j + " " + strings[i + j].toUpperCase());
            case 2: System.out.println("j = " + j + " " + strings[i - j].toLowerCase()); break;
            case 3: System.out.println("j = " + j + " " + strings[i + j].toUpperCase());
            case 4: System.out.println("j = " + j + " " + strings[i - j].toLowerCase()); break;
            case 5: System.out.println("j = " + j + " " + strings[i + j].toUpperCase());
            case 6: System.out.println("j = " + j + " " + strings[i - j].toLowerCase()); break;
            case 7: System.out.println("j = " + j + " " + strings[i + j].toUpperCase());
            case 8: System.out.println("j = " + j + " " + strings[i - j].toLowerCase()); break;
            default:
               System.out.println(strings[i - 7].toUpperCase());
         } // end switch
      } catch (Exception e) {
         System.out.println("Something Wrong!!!");
      } // end try
   } // end for
}
```

4. (20 Points) When the following program is run, it produces the given output. Write the rotate function. You may use Eclipse if you wish.

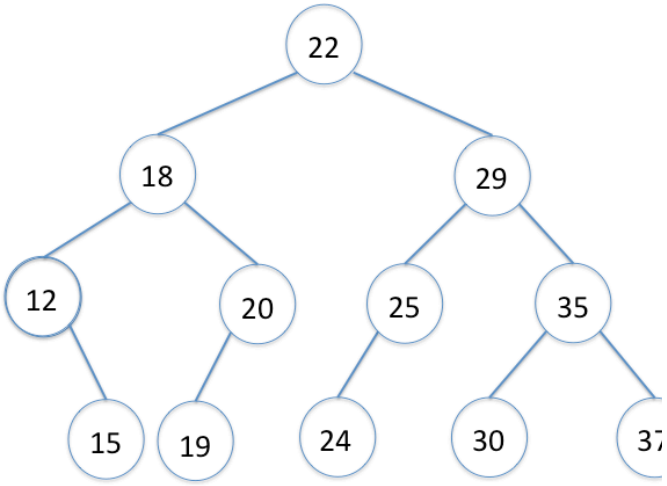
```
public class Q4 {

   public static String rotate(String s, int r) {    }

   public static void main(String[] args) {
         String name = "Sameh";
         for ( int i = 0 ; i <= name.length() ; i++ ) {
               System.out.println("When i = " + i + " - " + rotate(name,i));
         } // end for
   }
}
```

Output
When i = 0 - Sameh
When i = 1 - hSame
When i = 2 - ehSam
When i = 3 - mehSa
When i = 4 - amehS
When i = 5 - Sameh

5. (10 Points) Given that the following numbers are inserted, in the given order, into a binary tree. Draw the resulting tree.

(25, 19, 22, 15, 22, 16, 13, 11, 29, 27, 14, 21, 25, 24, 27, 28, 26, 36, 22, 18)

6. (20 Points) Given the following binary tree containing **int** s:



   a. (10 Points) What is the output of the following recursive method if it is initially called with the root node as a parameter:

```
public void preOrder(TreeNode n) {
   if (n != null) {
      System.out.print(n.getInt() + "  ");
      preOrder(n.getLeftChild());
      preOrder(n.getRightChild());
   } // end if
} // end preOrder
```

   b. (10 Points) What is the output of the following recursive method if it is initially called with the root node as a parameter:

```
public void postOrder(TreeNode n) {
   if (n != null) {
      postOrder(n.getLeftChild());
      postOrder(n.getRightChild());
      System.out.print(n.getInt() + "  ");
   } // end if
} // end postOrder
```

7. (10 Points) Use Eclipse to write the class **QueueEntry**. This class will have the following private attributes:

- *previous* – A pointer to the previous **QueueEntry**.
- *object* – A pointer to an **Object**.
- *next* – A pointer to the next **QueueEntry**.

Your **QueueEntry** class should have the following public methods:

- QueueEntry() - A constructor that initializes all attributes to null.
- QueueEntry(Object *object*) - A constructor that sets the object attribute to the specified *object* and the other attributes to null.
- get & set methods for all the attributes.

8. (20 Points) Use Eclipse to write the class **`Queue`**. This class will have the following private attributes:

   - *`head`* - A pointer to the first **`QueueEntry`** in the queue.
   - *`tail`* - A pointer to the last **`QueueEntry`** in the queue.

   Your **`Queue`** class should have the following public methods:

   - Queue() - A constructor that initializes the Queue to contain no entries.
   - Queue(QueueEntry *`entry`*) - A constructor that initializes the Queue to contain the specified *`entry`*.
   - Queue(QueueEntry[] *`entries`*) - A constructor that initializes the Queue to contain all the specified *`entries`*.
   - addEntry(QueueEntry *`entry`*) - A method that adds the specified *`entry`* to the back of the queue.
   - addEntries(QueueEntry[] *`entries`*) - A method that adds the specified *`entries`* to the queue in the same order as they appear in the array.
   - getFirstEntry() - A method that returns the first QueueEntry from the Queue. This method will remove the entry from the queue as well.
   - getHead() - Returns the head of the queue.
   - getTail() - Returns the tailg of the queue.
   - isEmpty() - A method that returns true if the Queue is empty, false otherwise.