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
	}

}
```

notB
ELSE
A
A

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);
    }
}
```

count = 3

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
}
```

```
j = 3 julio
Something Wrong!!!
Something Wrong!!!
j = 4 kalambay
Something Wrong!!!
Something Wrong!!!
j = 5 lisa
Something Wrong!!!
Something Wrong!!!
Something Wrong!!!
j = 7 ROBERT
Something Wrong!!!
LUCAS
```

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) {
         String rs = s.substring(s.length() - r);
         rs = rs.concat(s.substring(0, s.length() - r));
         return rs;
   }

   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
   }
}
```

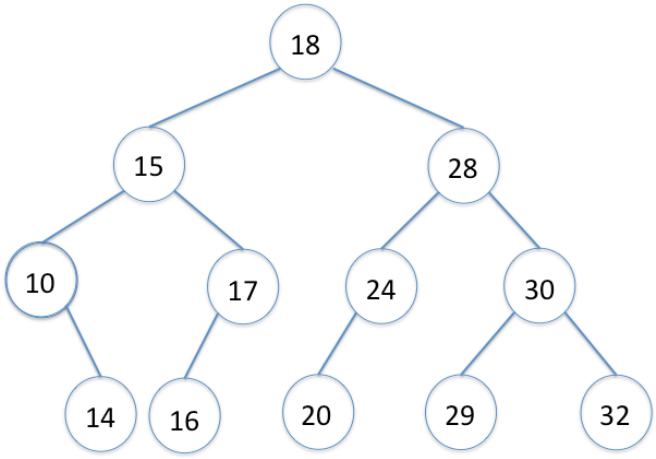
Output
When i = 0 - Sameh
When i = 1 - hSame
When i = 2 - ehSam
When i = 3 - mehSa
When i = 4 - amehS
When i = 5 - Sameh

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)

19 is the root
25 is the right child of 19
15 is the left child of 19
22 is the left child of 25
13 is the left child of 15
16 is the right child of 15
29 is the right child of 25
11 is the left child of 13
14 is the right child of 13
17 is the right child of 16
35 is the right child of 29
21 is the left child of 22
24 is the right child of 22
27 is the left child of 29
28 is the right child of 27
26 is the left child of 27
36 is the right child of 35
22 is the left child of 24
18 is the right child of 17
15 is the left child of 16

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
```

18 15 10 14 17 16 28 24 20 30 29 32

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
```

14 10 16 17 15 20 24 29 32 30 28 18

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.

```
public class QueueEntry {

	private QueueEntry previous, next;
	private Object object;

	public QueueEntry() {
		this(null, null, null);
	}

	public QueueEntry(Object object) {
		this(null, object, null);
	}

	private QueueEntry(QueueEntry previous, Object object, QueueEntry next) {
		this.previous = previous;
		this.object   = object;
		this.next     = next;
	}

	public QueueEntry getPrevious() {
		return previous;
	}

	public void setPrevious(QueueEntry previous) {
		this.previous = previous;
	}

	public QueueEntry getNext() {
		return next;
	}

	public void setNext(QueueEntry next) {
		this.next = next;
	}

	public Object getObject() {
		return object;
	}

	public void setObject(Object object) {
		this.object = object;
	}
}
```

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.

```
public class Queue {
	QueueEntry head, tail;

	public Queue() {
		this.head = null;
		this.tail = null;
	}
	public Queue(QueueEntry entry) {
		this.head = entry;
		this.tail = entry;
		entry.setPrevious(null);
		entry.setNext(null);
	}
	public Queue(QueueEntry[] entries) {
		int last = entries.length - 1;
		head = entries[0];
		entries[0].setPrevious(null);
		tail = entries[last];
		entries[last].setNext(null);
		for ( int i = 1 ; i <= last ; i++ ) {
			entries[i-1].setNext(entries[i]);
			entries[i].setPrevious(entries[i-1]);
		} // end for
	}
	public void addEntry(QueueEntry entry) {
		if (head == null) {
			this.head = entry;
			this.tail = entry;
			entry.setPrevious(null);
			entry.setNext(null);
		} else {
			tail.setNext(entry);
			entry.setPrevious(tail);
			entry.setNext(null);
			tail = entry;
		} // end if
	}
```

```
	public void addEntries(QueueEntry[] entries) {
		int last = entries.length - 1;
		if (head == null) {
			this.head = entries[0];
			entries[0].setPrevious(null);
			this.tail = entries[last];
			entries[last].setNext(null);
		} else {
			tail.setNext(entries[0]);
			entries[last].setPrevious(tail);
			entries[last].setNext(null);
			tail = entries[last];
		} // end if

		for ( int i = 1 ; i <= last ; i++ ) {
			entries[i-1].setNext(entries[i]);
			entries[i].setPrevious(entries[i-1]);
		} // end for
	}

	public QueueEntry getFirstEntry() {
		if (isEmpty()) {
			return null;
		} // end if

		QueueEntry entry = head;

		// only one entry?
		if (head == tail) {
			head = null;
			tail = null;
			return entry;
		} // end if

		head = entry.getNext();
		entry.getNext().setPrevious(null);

		return entry;
	}

	public QueueEntry getHead() {
		return head;
	}

	public QueueEntry getTail() {
		return tail;
	}

	public boolean isEmpty() {
		boolean answer = false;
		if (head == null) {
			answer = true;
		} // end if
		return answer;
	}
}
```

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
	}
}
```

notA
B
ELSE
B

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);
    }
}
```

count = 9

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
}
```

```
Something Wrong!!!
j = 3 SAMUEL
Something Wrong!!!
Something Wrong!!!
j = 4 ARIEL
Something Wrong!!!
j = 4 robert
Something Wrong!!!
j = 5 randy
Something Wrong!!!
j = 6 lucas
LUCAS
```

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) {
         String rs = s.substring(s.length() - r);
         rs = rs.concat(s.substring(0, s.length() - r));
         return rs;
   }

   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
   }
}
```

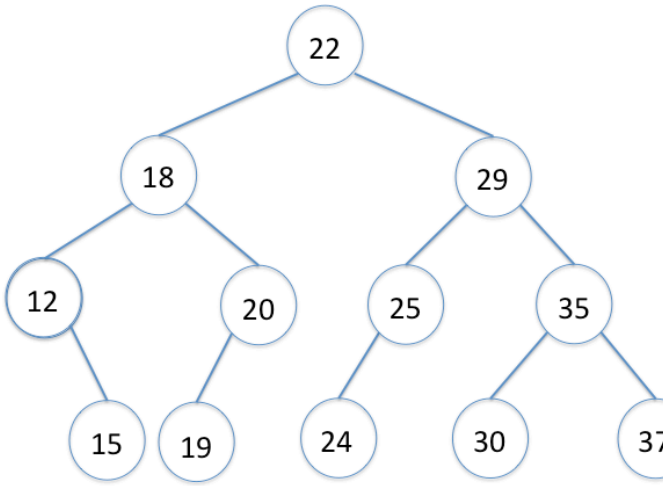
Output
When i = 0 - Sameh
When i = 1 - hSame
When i = 2 - ehSam
When i = 3 - mehSa
When i = 4 - amehS
When i = 5 - Sameh

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)

25 is the root
19 is the left child of 25
22 is the right child of 19
15 is the left child of 19
22 is the right child of 22
16 is the right child of 15
13 is the left child of 15
11 is the left child of 13
29 is the right child of 25
27 is the left child of 29
14 is the right child of 13
21 is the left child of 22
25 is the left child of 27
24 is the right child of 22
27 is the right child of 27
28 is the right child of 27
26 is the right child of 25
36 is the right child of 29
22 is the left child of 24
18 is the right child of 16

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
```

22 18 12 15 20 19 29 25 24 35 30 37

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
```

15 12 19 20 18 24 25 30 37 35 29 22

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.

```
public class QueueEntry {

   private QueueEntry previous, next;
   private Object object;

   public QueueEntry() {
          this(null, null, null);
   }

   public QueueEntry(Object object) {
          this(null, object, null);
   }

   private QueueEntry(QueueEntry previous, Object object, QueueEntry next) {
          this.previous = previous;
          this.object   = object;
          this.next     = next;
   }

   public QueueEntry getPrevious() {
          return previous;
   }

   public void setPrevious(QueueEntry previous) {
          this.previous = previous;
   }

   public QueueEntry getNext() {
          return next;
   }

   public void setNext(QueueEntry next) {
          this.next = next;
   }

   public Object getObject() {
          return object;
   }

   public void setObject(Object object) {
          this.object = object;
   }
}
```

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.

```
public class Queue {
	QueueEntry head, tail;

	public Queue() {
		this.head = null;
		this.tail = null;
	}
	public Queue(QueueEntry entry) {
		this.head = entry;
		this.tail = entry;
		entry.setPrevious(null);
		entry.setNext(null);
	}
	public Queue(QueueEntry[] entries) {
		int last = entries.length - 1;
		head = entries[0];
		entries[0].setPrevious(null);
		tail = entries[last];
		entries[last].setNext(null);
		for ( int i = 1 ; i <= last ; i++ ) {
			entries[i-1].setNext(entries[i]);
			entries[i].setPrevious(entries[i-1]);
		} // end for
	}
	public void addEntry(QueueEntry entry) {
		if (head == null) {
			this.head = entry;
			this.tail = entry;
			entry.setPrevious(null);
			entry.setNext(null);
		} else {
			tail.setNext(entry);
			entry.setPrevious(tail);
			entry.setNext(null);
			tail = entry;
		} // end if
	}
```

```
	public void addEntries(QueueEntry[] entries) {
		int last = entries.length - 1;
		if (head == null) {
			this.head = entries[0];
			entries[0].setPrevious(null);
			this.tail = entries[last];
			entries[last].setNext(null);
		} else {
			tail.setNext(entries[0]);
			entries[last].setPrevious(tail);
			entries[last].setNext(null);
			tail = entries[last];
		} // end if

		for ( int i = 1 ; i <= last ; i++ ) {
			entries[i-1].setNext(entries[i]);
			entries[i].setPrevious(entries[i-1]);
		} // end for
	}

	public QueueEntry getFirstEntry() {
		if (isEmpty()) {
			return null;
		} // end if

		QueueEntry entry = head;

		// only one entry?
		if (head == tail) {
			head = null;
			tail = null;
			return entry;
		} // end if

		head = entry.getNext();
		entry.getNext().setPrevious(null);

		return entry;
	}

	public QueueEntry getHead() {
		return head;
	}

	public QueueEntry getTail() {
		return tail;
	}

	public boolean isEmpty() {
		boolean answer = false;
		if (head == null) {
			answer = true;
		} // end if
		return answer;
	}
}
```