T/F Questions

1) **T**/F - protected methods can be accessed by classes in the same package and child classes
2) T/**F** - You can instantiate interfaces like Lists
3) T/**F** - A child class must call `super()` in the construtor
4) T/**F** - You can both extend multiple classes and implement multiple interfaces.
5) T/**F** - Interfaces can have instance variables

Multiple Choice

6) `public void question6(int i, int j)`
Possible method headers which will overload this method are:

a) private void question6(int i, int j){...}
b) public void question6(int i){...}
c) public boolean question6(int i, int b){...}
d) public void question6(int i, int b){...}

7) Fill in the blank condition for the following link list code that inserts an element at the end of a list.

```
public class Node{
	Object o;
	Node next;
	public Node(Object obj){
		O = obj;
		next = null;
	}
	public void setNext(Node n){
		next = n;
	}
}
	void insert(Node e){
		Node c = start
		while(/*Fill in blank*/){
			/*Fill in blank*/
		}
		c.next = e
	}
```

a) “c != null” and “c = c.next;”
b) “c != null” and “c.next = c;”
c) “c.next != null” and “c = c.next;”

d) “c.next != null” and “c.next = c;”

8) Consider the following class declarations;

public class IntList extends myList{...}
public class myList{...}

y is defined as:
IntList x = new IntList();
Object y = x;

Which of the following statements compile?

a)IntList a = (myList)y
b) IntList a = (IntList)y
c) y.toString()
d) myList a = (IntList)y

9)

public interface BasicLinkedList<K>{
public void Add(int index, K value){...}
public K Remove(int index){...}
…
}

A user creates an object of type K, and stores it in “BasicLinkedList<String> basLinLis”. Which of the following will compile?

a)String s = basLinLis.remove(-1);
b)basLinLis.add(0, null);
c)basLinLis.add(1, new StringBuilder(“Hello World!”))
d)Object o = basLinLis;

10) Which of the following modifiers is not allowed in abstract class method declarations?

a) final
b) static
c) private
d) Default

11) In the following question two classes are used. Class A has a method called p() which returns the string “A”. Class B inherits from A and overrides the p() method so that it now returns the string “B”. What is the output of the following code

```
A[] x = new A[5];
```

```
for(int i = 0; i < x.length; i++){
    if(i % 2 == 0){
        x[i] = new A();
    }else{
        x[i] = new B();
    }
}

System.out.print(x[0].p());
for(int i = 1; i < x.length; i++){
    System.out.print("," + x[i].p());
}
```

a) B,A,B,A,B
b) B,B,B,B,B
c) A,A,A,A,A
d) A,B,A,B,A

12) If a generic is declared in the class header it means

a) It means that all of the variables declared as the generic type are guaranteed to be the same type
b) The generic type is the only type that is allowed to be used in the class
c) Subclasses cannot have a specific type
d) There can only be one generic in the class

Short Answer

```
interface Animal {
            public String speak();
            public int legs();
            public String color();
      }

      class Dog implements Animal {
            public String speak() {
                  return "bark";
            }
            public int legs() {
                  return 4;
            }
            public String color() {
                  return "black";
            }
      }

      class Puppy extends Dog {
            public String speak(String teach) {
                  return teach;
            }

            public String color() {
                  return "brown";
            }
      }
```

13) show the output when running:

```
Animal a = new Dog();
System.out.println(a.legs());
```

Answer: 4

14)show the output when running:

```
Animal a = new Puppy();
System.out.println(a.speak());
```

Answer: bark

15)show the output when running:

```
Animal a = new Puppy();
System.out.println(a.color());
```

Answer: brown

16) **Part 1**

Complete the compareTo function based on the student's id number.

```
public class Student implements Comparable<Student> {
	private int id;

	public Student(int id) {
		this.id = id;
	}

	public int compareTo(Student other) {
		//student implement
		Return this.id - other.id;
	}
}
```

Part 2

Students should first be compared based on the length of their names. If they have the same name, then they need to be compared alphabetically.

```
public class Student implements Comparable<Student> {
	private int id;
	private String name;
	public Student(int id, String name) {
	  this.id = id;
	  this.name = name;
	}

	public int compareTo(Student other) {
		//Finish the rest
		if (this.name.length() < other.name.length()) { return -1; }
		else if (this.name.length() > other.name.length()) { return 1; }
		else { return this.name.compareTo(other.name); }
	}
}
```

17) given the linked list
myList = [“a”] -> [“b”] -> [“c”] -> [“d”] -> [“e”] -> [“f”] -> [“g”] -> [“h”]
First points to the first Node ([“a”])
And the class:

```
class Node {
	String data;
	Node next;
}
```

What will be the resulting linked list after the following code is executed?

```
Node curr = first;
While (curr != null && curr.next != null) {
	curr.next = curr.next.next;
	curr = curr.next;
}
```

myList = [“a”] -> [“c”] -> [“e”] -> [“g”]

18) Remember the LoopBag discussion. Instead of a linked list, we have an Array Implemented Bag. Here is the incomplete intersect function:

```
public void intersect(LoopBag<E> lb) {
	Iterator<E> iter = lb.iterator();
	E[] temp = (E[]) new Object[this.capacity];
	int position = 0;
	for (int i = 0; i < lb.size(); i++) {
		E item = iter.next();
		if (contains(item)) {
			temp[position++] = item;
		}
	}
	items = temp;
}
```

You can assume loopbags have a public boolean contains(E item) method which returns true if the item is found in the loopbag. Fill in the missing code, taking into account that a loopbag uses an array and not a linked list. You can use the iterator iter to iterate through the list.

19) what function needs to be added when a class implements Comparable and explain what is returned and why,what is passed in, and how the function works.
-> it needs to override the int compareTo(Object o) function.
->The compareTo function returns an integer and takes an object as anargument
->The function will return an positive, negative, or 0 integer according to which object comes first. If this. Object is less than the parameter object, it will return a negative number. If this. Object is larger than the parameter then it will return positive, and it will return 0 if both of the objects are equal.

20) Explain the difference between an abstract class and an interface and explain why one would want to use one over the other

21) What is a benefit of using a linked list over an array? What is a benefit of using an array over a linked list? If you had to make an ArrayList, is an array or linked list better?

Debugging

22)
Here is a working setOrigin():

```
public final boolean setOrigin(Coordinate co) {
        ArrayList<Coordinate> temp = new ArrayList<>();
        for(Coordinate c: layout[orientation])
            temp.add(co.translate(c.col, c.row));

        for(Coordinate c: temp) {
            if (c.row >= game.getMaxRows() || c.row < 0 ||
                c.col >= game.getMaxCols() || c.col < 0)
                return false;
            if (!(locations.contains(c) ||
                game.getBoardCell(c.col, c.row) == cell.EMPTY))
                return false;
        }
        for (Coordinate c: locations)
          game.setBoardCell(c.col, c.row, cell.EMPTY);
        locations = temp;
        for (Coordinate c: locations)
          game.setBoardCell(c.col, c.row, cell);
        return true;
    }
```

Here is a buggy rotate() in I.java.

```
public I(Game game) {
    super(game, "I", Cell.CYAN);
```

```
    ArrayList<Coordinate> vertical = new ArrayList<>();
    ArrayList<Coordinate> horizontal = new ArrayList<>();
    vertical.add(new Coordinate(0,0));
    vertical.add(new Coordinate(0,1));
    vertical.add(new Coordinate(0,2));
    vertical.add(new Coordinate(0,3));
    horizontal.add(new Coordinate(0,0));
    horizontal.add(new Coordinate(1,0));
    horizontal.add(new Coordinate(2,0));
    horizontal.add(new Coordinate(3,0));
    layout[0] = vertical;
    layout[2] = vertical;
    layout[1] = horizontal;
    layout[3] = horizontal;
  }

  @Override
  public boolean rotate() {
    Coordinate c = layout[-1].get(0); //ERROR 1
      if (orientation == 0 || orientation ==2)
          c = getOrigin().translate(-2,1);
    else
      c = getOrigin().translate(1,-1);

  orientation++; //ERROR 2
  if (!setOrigin(c)){
    return false; //ERROR 3
  }
  return true;
  }
```

What is wrong here and how should you fix it?
Problem: **layout[-1].get(0); Indexout of bounds exception/NullPointer**
Fix: **Just set it to null or something**
Problem: orientation++; needs a top bound
Fix: **if (++orientation > 3) orientation = 0;**
Problem: **if setting the origin does not work, you need to decrement orientation**
Fix: **if (--orientation < 0) orientation = 3;**

```
23) Here is code:
Public class LinkedListWithErrors<E>{
  Node<E> head;
```

```
  public boolean add(int index, E value){
    Object current = head; //ERROR 1
    int curInd = 0;
    if(index == 0){
      head = new Node(value, null); //ERROR 2
      return true;
    }

    while (current != null){
       if (index == curInd){
         Node<E> nNode = new Node<E>(value, null);
         current.next = nNode; //ERROR 3
         nNode.next = current.next;
         return true;
       }
       current = current.next
       curInd++;
    }
    return false;
  }

  public String get(int index){ //ERROR 4
    Node<E> current = head;
    int curInd = 0;
    while (curInd < index){
        current = current.next //ERROR 5
        curInd++;
    }
    return current.value;
  }
}
```

Identify 5 problems and give a fix for each
Problem: **Object has no access to node methods**
Fix: **Node Current =...**

Problem: **Potentially lose the rest of the list**
Fix: **head = new Node(value, head)**

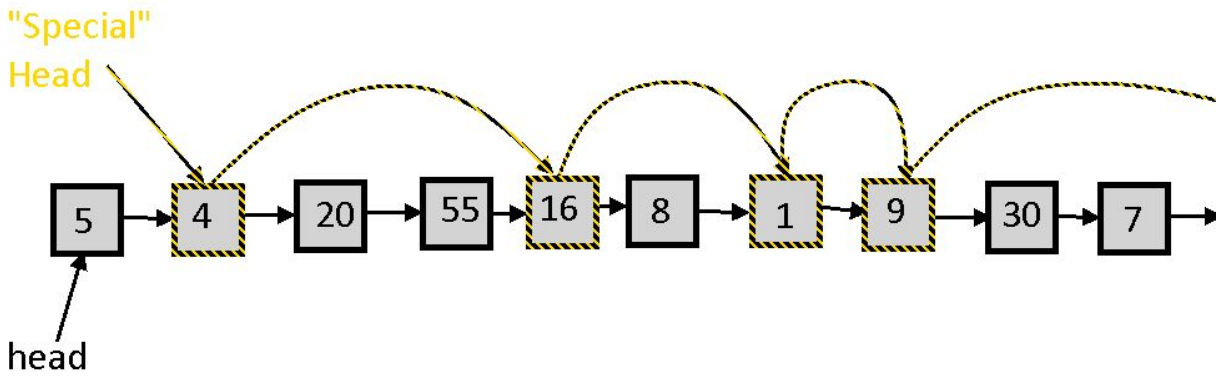
Problem: **Circular list creation**
Fix: **swap line with one below it**

Problem: **Wrong return type**
Fix: **public E...**

Problem: **NullPointerException**
Fix: **while((curInd < index) && (current != null))... if (current==null) {return null};**

Programming

24) linked list?
We need to make a class known as SpecialLinkedList. Most nodes will have only one element, but a select handful of elements- “Special” elements, if you will- are going to be accessed a lot, and therefore each “special” element has a pointer to the next “special” element. See image below, yellow stripes means the node is special:



The node class and interface you will be using look like so:

```
public class Node<E>{

    public E data;
    public Node next;
    public Node nextSpec;
    public boolean isSpecial;

    public Node(E data, Node next, Node nextSpec, boolean isSpecial){
    this.data = data;
    this.next = next;
    this.nextSpec = nextSpec;
    this.isSpecial = isSpecial;
    }
}
```

```
interface specialListInterf<E>{

    //adds a regular element to the list. First element is position 0.
    public boolean add(E data);

    //adds a special element to the list at position index.
    //first element is position 0.
    public boolean addSpecial(E data);

    //returns the data field for the element at position index.
    //does not matter if this element is special or not
    //returns null if it does not exist.
    public E get(int index);

    //returns the data field for the nth special element.
    //returns null if it does not exist.
    //ignores all non-special elements when counting.
    public E getSpecial(int specIndex);
}
```

So, implement the following class:

```
Public class specialLinkedList<E> extends specialListInterf<E>{
      Node<E> head = null
      Node<E> specialHead = null;
      public boolean add(E data){
            Node<E> nNode = new Node<>(data, null, null, false);
            if(head == null){
                  head = nNode;
                  return true;
            }
            Node<E> curr = head;
            while(curr.next!=null){
                  curr = curr.next;
            }
            curr.next = nNode;
            return true;
      }

      public boolean addSpecial(E data){
            Node<E> nNode = new Node<>(data, null, null, true);
            if(head == null){
                  head = nNode;
                  specialHead = nNode;
```

```
                return true;
            }
            Node<E> curr = head;
            Node<E> lastSpecial = specialHead;
            while(curr.next!=null){
                if (curr.isSpecial)
                    lastSpecial = curr;
                curr = curr.next;
            }
            curr.next = nNode;
            if(specialHead == null)
                specialHead = nNode;
            else
                lastSpecial.nextSpec = nNode;
            return true;
        }
        public E get(int index){
            for(Node<E> curr = head; curr!= null; curr = curr.next){
                if (index == 0)
                    return curr;
                index--;
            }
            return null;
        }
        public E getSpecial(int specIndex){
        for(Node<E> curr = specialHead; curr!= null; curr = curr.nextSpec){
            if (index == 0)
                return curr;
            index--;
        }
        return null;
}
```

25)

```
public class IntList implements Iterable{
      ArrayList<Integer> myList = new Arraylist<>();
      public IntList(){
            for(int i = 0; i < 100; i++){
                  myList.add((int) Math.floor(Math.random() * 101));
            }
      }
```

```
	public Iterator<Integer> iterator(){
		return new EvenIterator();
	}
	private class EvenIterator implements Iterator{
		int position;
		ArrayList<Integer> evens;
		public EvenIterator(){
			position = 0;
			evens = new Arraylist<>();
			for(Integer i: myList)
				if(i %2 ==0)
					evens.add(i);
		}
		public boolean hasNext(){
			return position < evens.size();
		}
		public Integer next(){
			return evens.get(position++);
		}
	}
}
```