# Lecture 12 -
# Makeup for WrittenTest1

# (≈ 90 minutes)

# Generic DLL in Java: Inserting to the Front/End

```
@Test
public void test_String_DLL_Insert_Front_End() {
    DoublyLinkedList<String> list = new DoublyLinkedList<>();
    list.addFirst("Mark");
    list.addFirst("Alan");

    assertTrue(list.getSize() == 2);
    assertEquals("Alan", list.getFirst().getElement());
    assertEquals("Mark", list.getFirst().getNext().getElement());

    list = new DoublyLinkedList<>();
    list.addLast("Mark");
    list.addLast("Alan");

    assertTrue(list.getSize() == 2);
    assertEquals("Alan", list.getLast().getElement());
    assertEquals("Mark", list.getLast().getPrev().getElement())
}
```

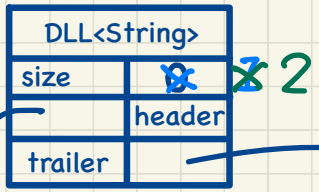
Exercise: Tracing

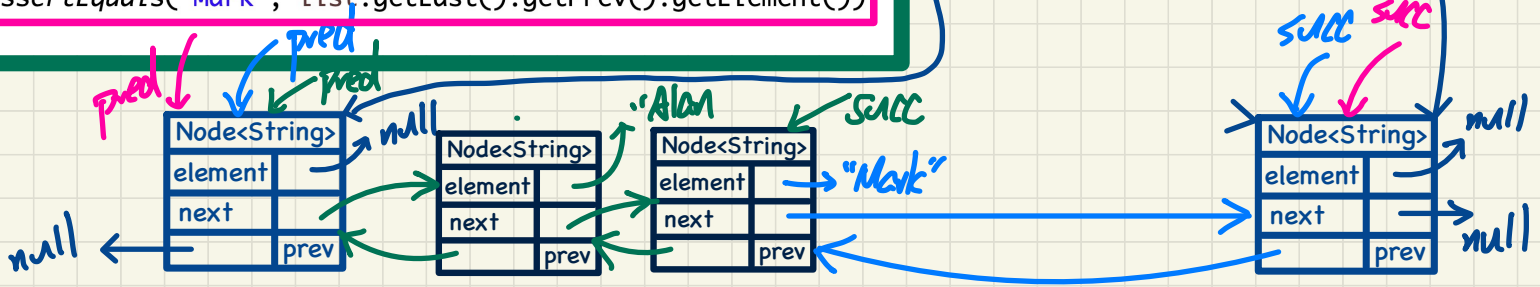
```
void addFirst(E e) {
    addBetween(e, header, header.getNext());
}
```

pred succ

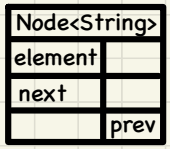
```
void addLast(E e) {
    addBetween(e, trailer.getPrev(), trailer);
}
```

pred succ



# Generic DLL in Java: Inserting to the Middle




```java
@Test
public void test_String_DLL_addAt() {
    DoublyLinkedList<String> list = new DoublyLinkedList<>();
    list.addAt(0, "Alan");
    list.addAt(1, "Tom");
    list.addAt(1, "Mark");

    assertTrue(list.getSize() == 3);
    assertEquals("Alan", list.getFirst().getElement());
    assertEquals("Mark", list.getFirst().getNext().getElement());
    assertEquals("Tom", list.getFirst().getNext().getNext().getElement());
}
```

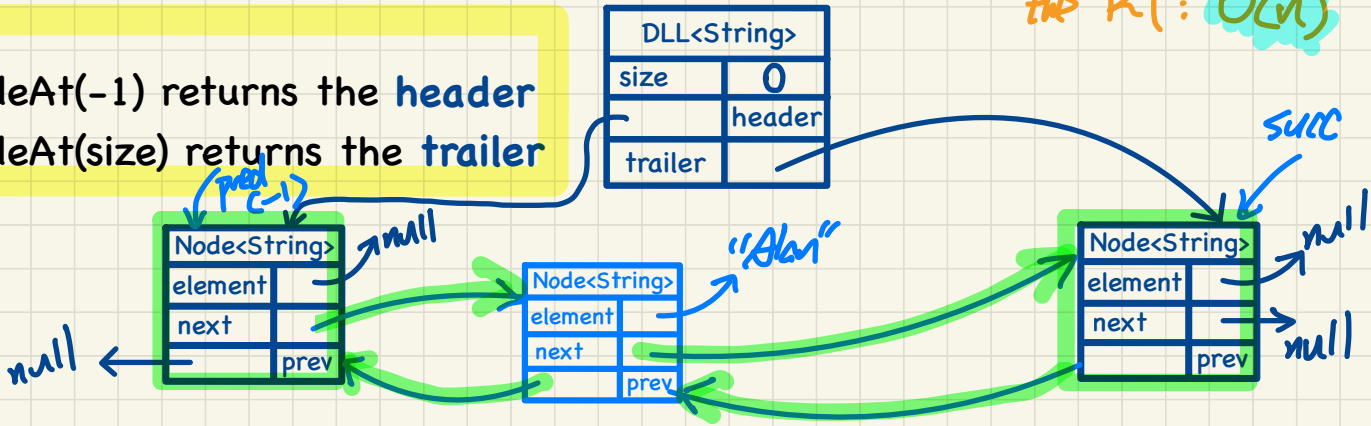
Exercise: Tracing

```java
addAt(int i, E e) {
  if (i < 0 || i > size) {
    throw new IllegalArgumentException
  else {
    Node<E> pred = getNodeAt(i - 1);
    Node<E> succ = pred.getNext();
    addBetween(e, pred, succ);
  }
}
```

still dominates the RT: O(n)

**Notes.**

+ getNodeAt(-1) returns the **header**
+ getNodeAt(size) returns the **trailer**



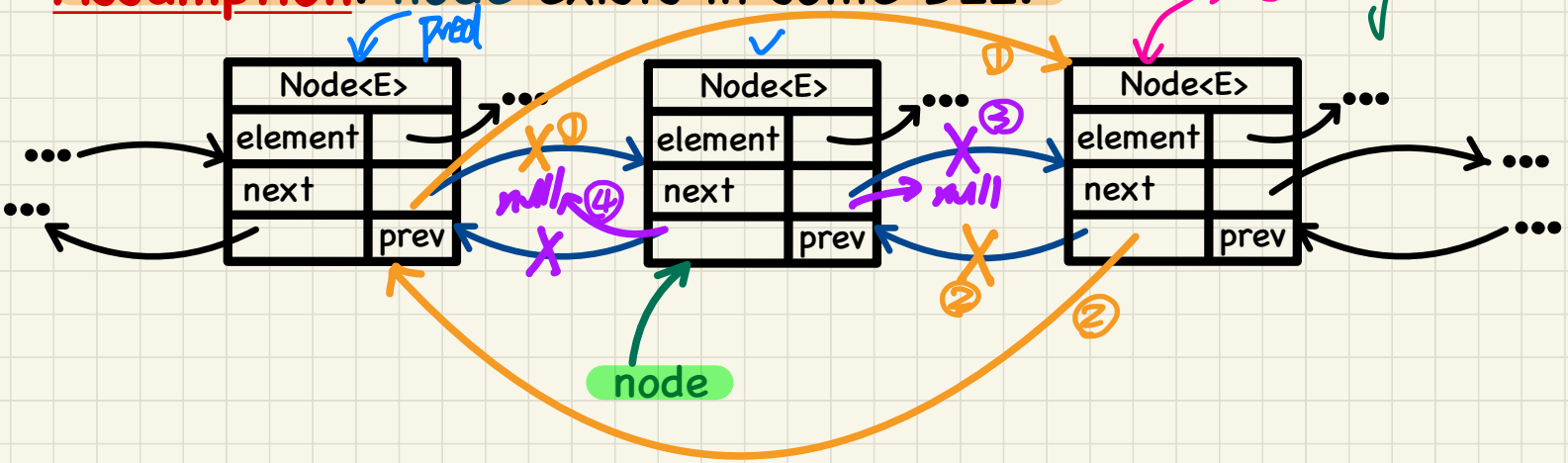
# Generic DLL in Java: Removing a Node

```
1  void remove (Node<E> node) {
2    Node<E> pred = node.getPrev();
3    Node<E> succ = node.getNext();
4    pred.setNext(succ);
5    succ.setPrev(pred);
6    node.setNext(null);
7    node.setPrev(null);
8    size --;
9  }
```

RT: O(1)

efficient solely because the ref. of the node to remove is given.

**Assumption**: node exists in some DLL.



# Generic DLL in Java: Removing from the Front/End

```java
@Test
public void test_String_DLL_Remove_Front_End() {
    DoublyLinkedList<String> list = new DoublyLinkedList<>();
    list.addFirst("Mark");
    list.addFirst("Alan");
    list.removeFirst();
    list.removeFirst();
    assertTrue(list.getSize() == 0);

    list = new DoublyLinkedList<>();
    list.addFirst("Mark");
    list.addFirst("Alan");
    list.removeLast();
    list.removeLast();
    assertTrue(list.getSize() == 0);
}
```

```java
void removeFirst() {
  if (size == 0) { throw new IllegalArgumentException("Empty"); }
  else { remove(header.getNext()); }
}
```

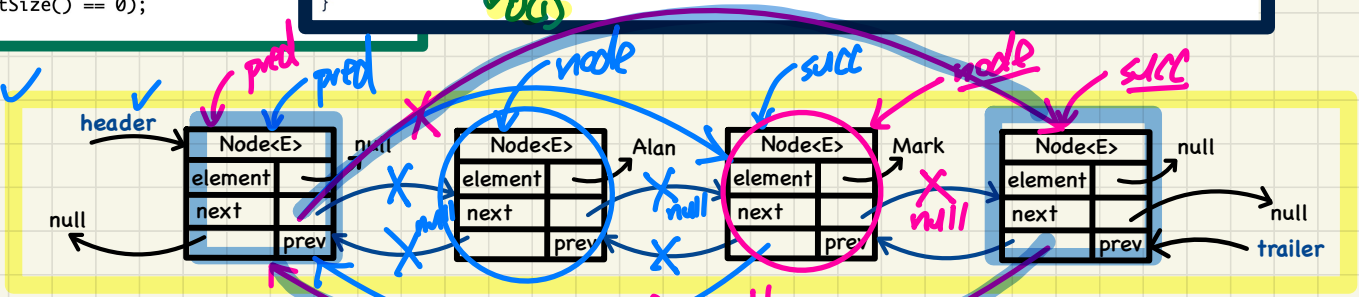
O(1)

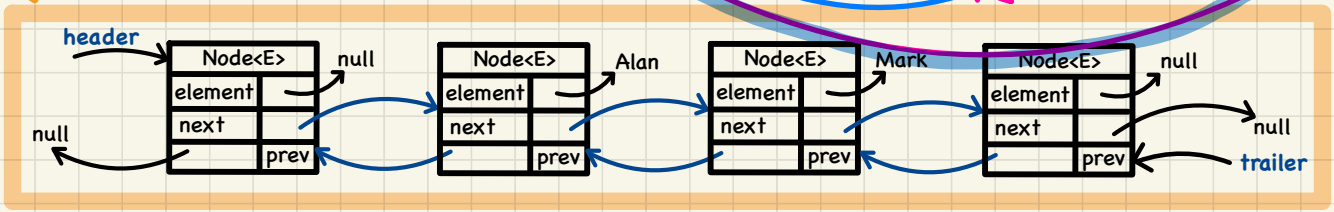
first node

```java
void removeLast() {
  if (size == 0) { throw new IllegalArgumentException("Empty"); }
  else { remove(trailer.getPrev()); }
}
```

O(1)

Exercise: Tracing



# Generic DLL in Java: Removing from the Middle

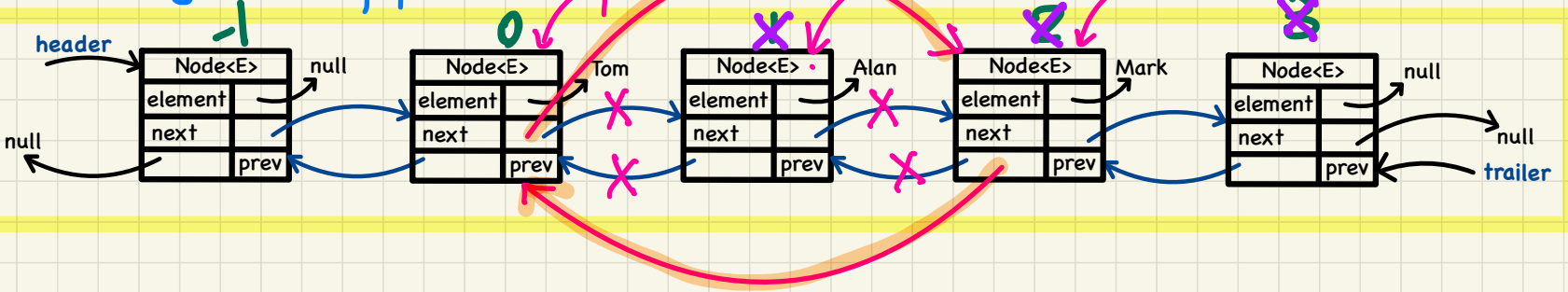
```java
@Test
public void test_String_DLL_removeAt() {
    DoublyLinkedList<String> list = new DoublyLinkedList<>();
    list.addFirst("Mark");
    list.addFirst("Alan");
    list.addFirst("Tom");
    assertTrue(list.getSize() == 3);
    list.removeAt(1);
    assertTrue(list.getSize() == 2);
    list.removeAt(0);
    assertTrue(list.getSize() == 1);
    list.removeAt(0);
    assertTrue(list.getSize() == 0);
}
```

```java
removeAt (int i) {
   if (i < 0 || i >= size) {
      throw new IllegalArgumentExce
   else {
      Node<E> node = getNodeAt(i);
      remove (node);
   }
}
```

dominates RT: O(n)

Contrast SLL removeAt: getNodeAt (i-1)

Exercise: Tracing (paper & Eclipse)



# Lecture 2

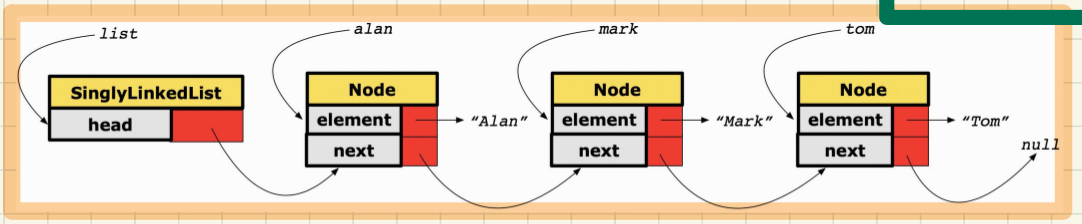
## Part K

## *Doubly-Linked Lists - Comparing Arrays, SLL, and DLL*

# Running Time: Arrays vs. SLL vs. DLL

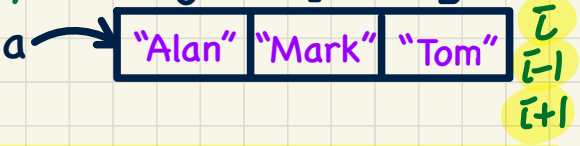
see discussion at the end of SLL.

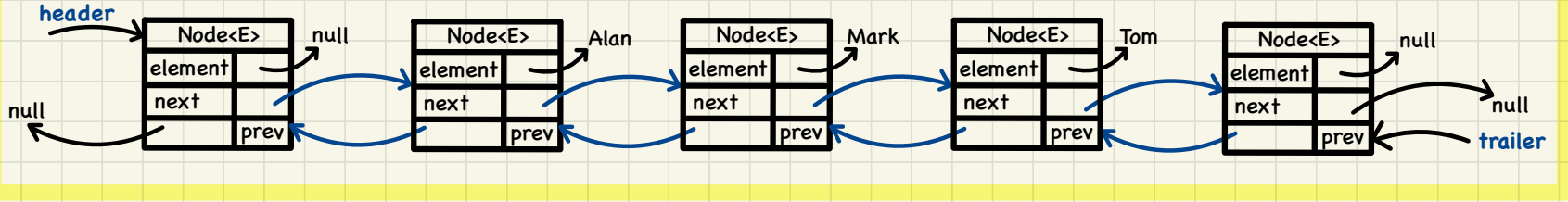
| Operation \ Data Structure | Array | Singly-Linked List | Doubly-Linked List |
|---|---|---|---|
| size | O(1) | | |
| first/last element | O(1) | | |
| element at index i | O(n) | O(n) | O(n) |
| remove last element | O(n) | O(1) remove at index i: | O(1) |
| add/remove first element, add last element | | | |
| add/remove $i^{th}$ element | given reference to $(i-1)^{th}$ element | | |
| | not given | | |




pass (i-1) node.

remove at index i:



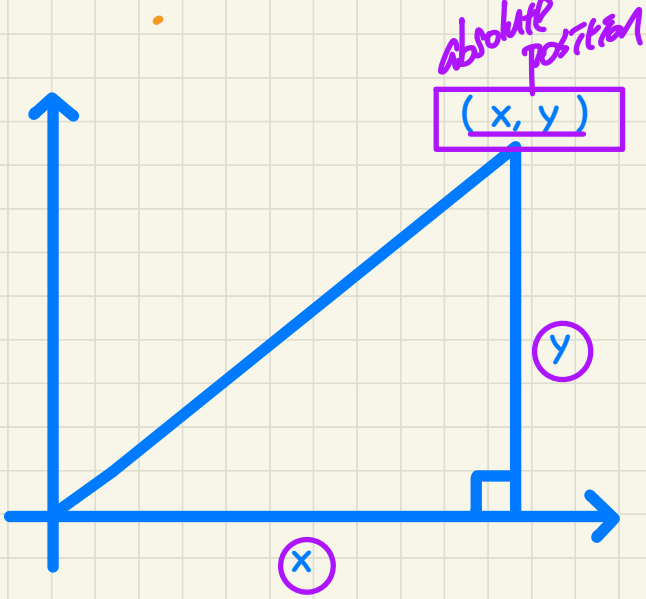
# Lecture

## Implementing ADT in Java

### *Interfaces*

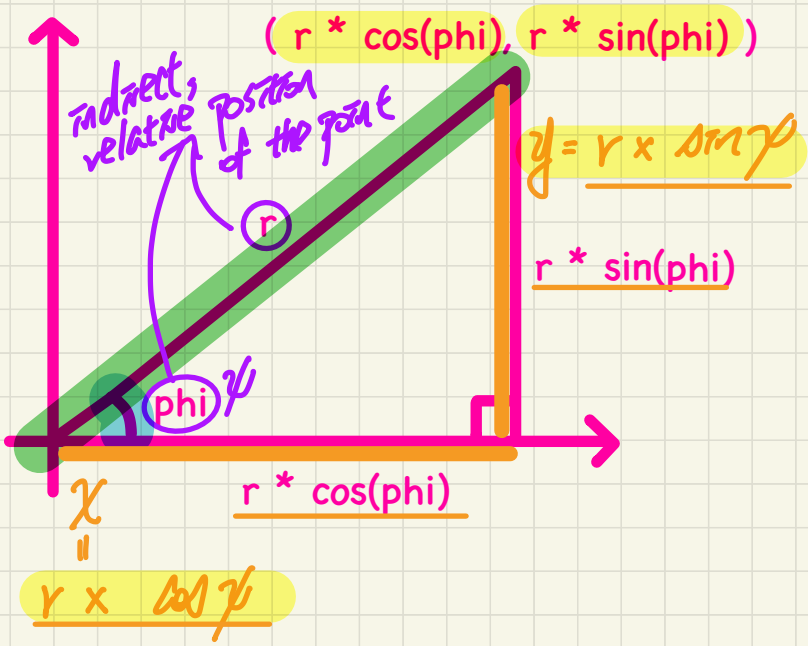
# Representations of 2-D Points: Cartesian vs. Polar

## Cartesian System



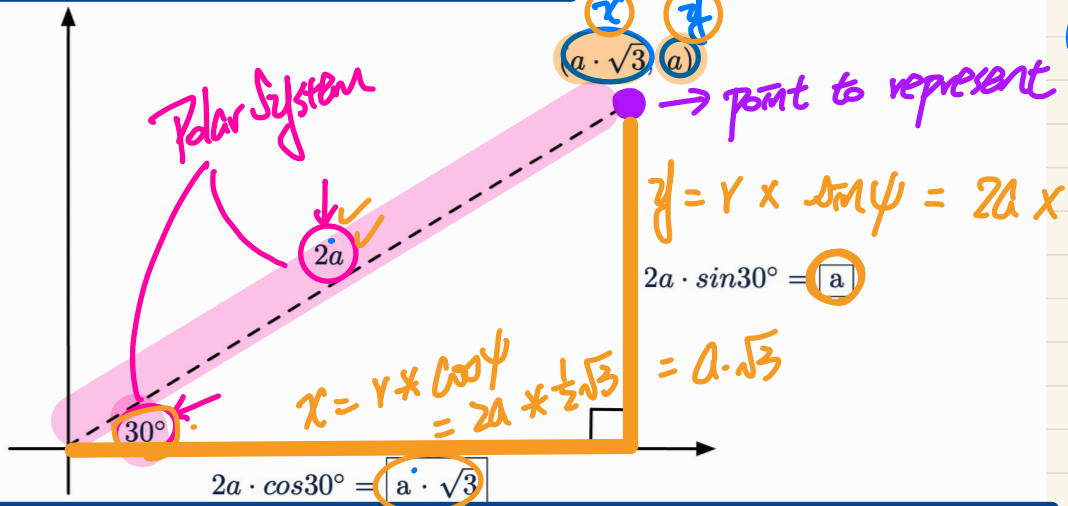

Goal: Dynamically, switch between two systems seamlessly.

## Polar System

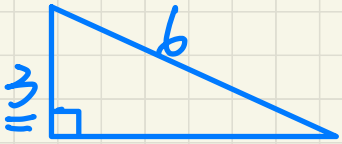


# Example: Cartesian vs. Polar

Recall: $sin30° = \frac{1}{2}$ and $cos30° = \frac{1}{2} \cdot \sqrt{3}$




$a = 3$




$3 \cdot \sqrt{3}$

$(3)^2 + (3 \cdot \sqrt{3})^2 = b^2$

We consider the same point represented differently as:

- $r = 2a$, $\psi = 30°$ [ polar system ]
- $x = 2a \cdot cos30° = a \cdot \sqrt{3}$, $y = 2a \cdot sin30° = a$ [ cartesian system ]

# Interface vs. Implementations

Interface used as a static type

```
double A = 5;
double X = A * Math.sqrt(3);
double Y = A;
Point p;
p = new CartisianPoint(X, Y); /* polymorphism */
print("(" + p.getX() + ", " + p.getY() + ")");
p = new PolarPoint(2 * A, Math.toRadians(30));
print("(" + p.getX() + ", " + p.getY() + ")");
```

static method.

DT: CartesianPoint

DT: PolarPoint

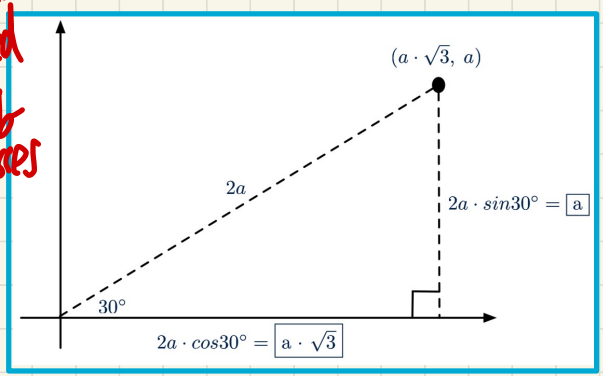
Point p = new Point(); ✗ not valid.

× p.getX() × p.getY()

| CartesianPoint | |
|---|---|
| x | $5\cdot\sqrt{3}$ |
| y | 5 |

Point p

| PolarPoint | |
|---|---|
| r | 10 |
| phi | 30° |




implementations deferred to sub classes

an abstract class where all methods are abstract

available across packages.

```
public interface Point {
    public double getX();
    public double getY();
}
```

headers of methods

forced to implement

implements

```
public class CartesianPoint implements Point {
    private double x;
    private double y;
    public CartesianPoint(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public double getX() { return x; }
    public double getY() { return y; }
}
```

absolute position

```
public class PolarPoint implements Point {
    private double phi;
    private double r;
    public PolarPoint(double r, double phi) {
        this.r = r;
        this.phi = phi;
    }
    public double getX() { return Math.cos(phi) * r; }
    public double getY() { return Math.sin(phi) * r; }
}
```

relative position

measured in radians.