Generics

The feature of Generics in Java is often referred to as parametric polymorphism in the literature. It is similar to the feature of templates in C++, but there are some fundamental differences.
public class Pair<T> {
    public T fst; public T snd;
    Pair (T x, T y) { fst= x ; snd= y ; }
    public static<T> // functional
        Pair<T> swap (Pair<T> p) {
            return new Pair<T> (p.snd, p.fst) ;
        }
}

public class Pair<T1,T2> {
    public T1 fst; public T2 snd;
    Pair (T1 x, T2 y) { fst= x ; snd= y ; }
    public static<T1,T2> // functional
        Pair<T2,T1> swap (Pair<T1,T2> p) {
            return new Pair<T2,T1> (p.snd, p.fst);
        }
}
public class Node<T> {
    T item ;
    Node<T> next ;
    Node () { } ;
    Node (T x) { item = x ; }
    Node (T x, Node<T> xs) { item = x ; next = xs ; }
    // ... (omitted code)
} // end of [Node]
public class Node<T> {
    // ... (omitted code)
    public static<T>
    int length (Node<T> xs) {
        int res = 0 ;
        while (xs != null) {
            xs = xs.next ; res = res + 1 ;
        }
        return res ;
    } // end of [length]
} // end of [Node]
public class Node<T> {
    // ... (omitted code)
    public static<T>
    Node<T> copy (Node<T> xs) {
        Node<T> ys0 = null;
        if (xs == null) return ys0;
        ys0 = new Node<T>(xs.item); xs = xs.next;
        Node<T> ys = ys0;
        // copying xs into ys.next
        while (xs != null) {
            ys.next = new Node<T>(xs.item);
            xs = xs.next; ys = ys.next;
        }
        ys.next = null; return ys0;
    } // end of [copy]
} // end of [Node]
Divide and Conquer

In essence, divide-and-conquer means to solve a problem by reducing it to similar problems of smaller sizes.
Hanoi Towers: description

There are three poles: left, middle and right, and there are $n$ disks of distinct sizes stacked at the left pole. A disk can be stacked on the top of another one only if the former is smaller than the latter. Please move all the disks for the left pole to the middle pole by moving one disk at each time.
Hanoi Towers: demo
Hanoi Towers: implementation

```c
static
void hanoi (int N, int d) { // d = 1 or -1
    if (N == 0) return ;
    hanoi (N-1, -d); shift (N, d); hanoi (N-1, -d)
    return ;
} // end of [hanoi]
```
Coin Change: description

Assume that there are unlimited number of coins of $n$ different denominations. Given a fixed amount, please find the total number of distinct ways that this amount can be expressed as the sum of a set of coins.
Coin Change: implementation

class CoinCHG {
    static int coinArr[] = { 1, 5, 10, 25 } ;
    private static int coinchg_ (int sum, int n) {
        if (sum == 0) return 1 ; else if (n == 0) return 0 ;
        int c = coinArr[n-1] ;
        if (c <= sum) {
            return coinchg_ (sum, n-1) + coinchg_ (sum-c, n) ;
        } else {
            return coinchg_ (sum, n-1) ;
        } // end of [if]
    } // end of [coinchg_]
    public static int coinchg (int sum) {
        return coinchg_ (sum, coinArr.length) ;
    } // end of [coinchg]
} // end of [CoinCHG]
A Monte Carlo approach to computing $\pi$

Let us see the implementation.
End of the slides for lecture 6