1. Declarations
   • Type ... the "nature" of the thing
   • Variable ... the symbolic location of a piece of data of a particular type

2. Simple types
   • Integer (int) 14
   • real (float, double) 14.3
   • char a
   • boolean true or false
   • defined group of symbols
     type taffe_kid(Bill, Pat, Tom, Marybeth, Terry, Chris)
   • pointer … an integer that points to a piece data … the address of the data
     &ptr ... the address itself
     *ptr ... the value that is at that address

3. Compound types
   • string “Bill”
   • array
     • an ordered list of the same type
     • indexed
     • array of integers ... array of string
       a[1], a[2], a[3] ... a[n]
     • record ... an ordered list of different types
       record
       age : integer
       salary : real
       last_name : string
       middle_initial : char
       alive : boolean
       end
   • set ... an unordered list of simple type ... things are either in a set or they aren’t in the set. We don’t ask “how many of this thing are in the set?”
4. Objects

- a compound data structure (like a record) plus the operations that can be done on it or the queries made of it ... called methods
- the general “thing” is called a class. Each instance of the “thing” is called an object
- classes/objects have three important properties:
  - **encapsulation** ... the combination of the data and operations and queries that can be done on it
  - **inheritance** ... sub-objects "inherit" all the data type and operations and queries of the "parent" object
  - **polymorphism** ... one operation can be performed on many data types

- Example ... in java-like pseudo code

```java
// java-like pseudo code ... some syntax is missing

// class Day
// class Day has two constructors
// class Day(int year, int month, int day)
// class Day();  sets day, month and year from system
// class Day has several methods including
// int getDay();  int getMonth();  int getYear();

class ManagerTest
{
    main (String[] args)
    {
        Employee[] staff = new Employee[3];

        staff[0] = new Employee("Harry Hacker", 35000,
                          new Day(1989, 10, 1));
        staff[1] = new Employee("Tony Tester", 38000,
                          new Day(1990, 12, 15));
        staff[2] = new Manager ("Carl Cracker", 75000,
                          new Day(1987, 12, 15));

        for (I = 0; I < 3; I++) staff[I].raiseSalary(5);
        for (I = 0; I < 3; I++) staff[I].print();
    } // end of class main
```
class Employee
{
    String name;
    Double salary;
    Day hireDay;

    Employee(String n, double s, Day d)
    {
        name = n;
        salary = s;
        hireDay = d;
    }

    print()
    {
        // printing statements for the class go here
    }

    raiseSalary(double byPercent)
    {
        salary = salary * (1 + byPercent/100);
    }

    hireYear()
    {
        return hireDay.getYear();
    }
} // end of class Employee
class Manager extends Employee
{
    String secretaryName;

    Manager(String n, double s, Day d)
    {
        super(n,s,d);
        secretaryName = " ";
    }

    raiseSalary(double byPercent)
    {
        Day today = new Day();
        double bonus = 0.5*(today.getYear() - hireYear());
        super.raiseSalary(byPercent + bonus);
    }

    setSecretaryName(String n)
    {
        secretaryName = n;
    }
} // class Manager
5. Linked list ... a group of nodes
   • node ... a piece of data (of any data type) with a pointer to the next node
   • methods ... insert and delete nodes, traverse the list
   • complex linked lists ... circularly linked lists, doubly linked lists

6. Stack ... a restricted access list
   • pointer to top of stack
   • push and pop
   • easy to reverse a list

7. Queue ... a restricted access list
   • pointers to front and rear
   • enter at rear
   • leave at front
   • circular queue

8. Tree ... a linked list of nodes each of which give rise to multiple sub-nodes
   all emanating from a single node

9. Interesting example ... Binary Search Tree (we saw this in algorithms)
   
   make first node
   while more_letters_exist
     repeat
       if letter <= node_letter take left branch
       if letter > node_letter take right branch
     until pointer is nil
   make new node

   To see sorted order ... traverse tree with inorder traversal
   in_order_traverse(ptr)
     if left_branch_not_nil
       in_order_traverse (left_branch)
     list node letter
     if right_branch_not_nil
       in_order_traverse (right_branch)