# Chapter 1

Introduction

# The purpose and contents of the course

- Introduce most used data structures and algorithms
- Prerequisite of other courses
- Introduce algorithm analysis
- Review Java and C++

# The purpose and contents of the course

1. Introduce most used data structures and algorithms.

   Use proper data structures to solve different problems.

Example :

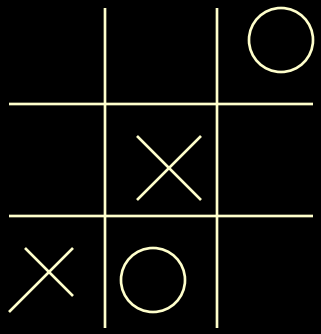
- Game problem
- Management of library catalogue by computer
- Management of the traffic lights in intersections
- The book : selection problem

  solve a popular word puzzle

# The purpose and contents of the course

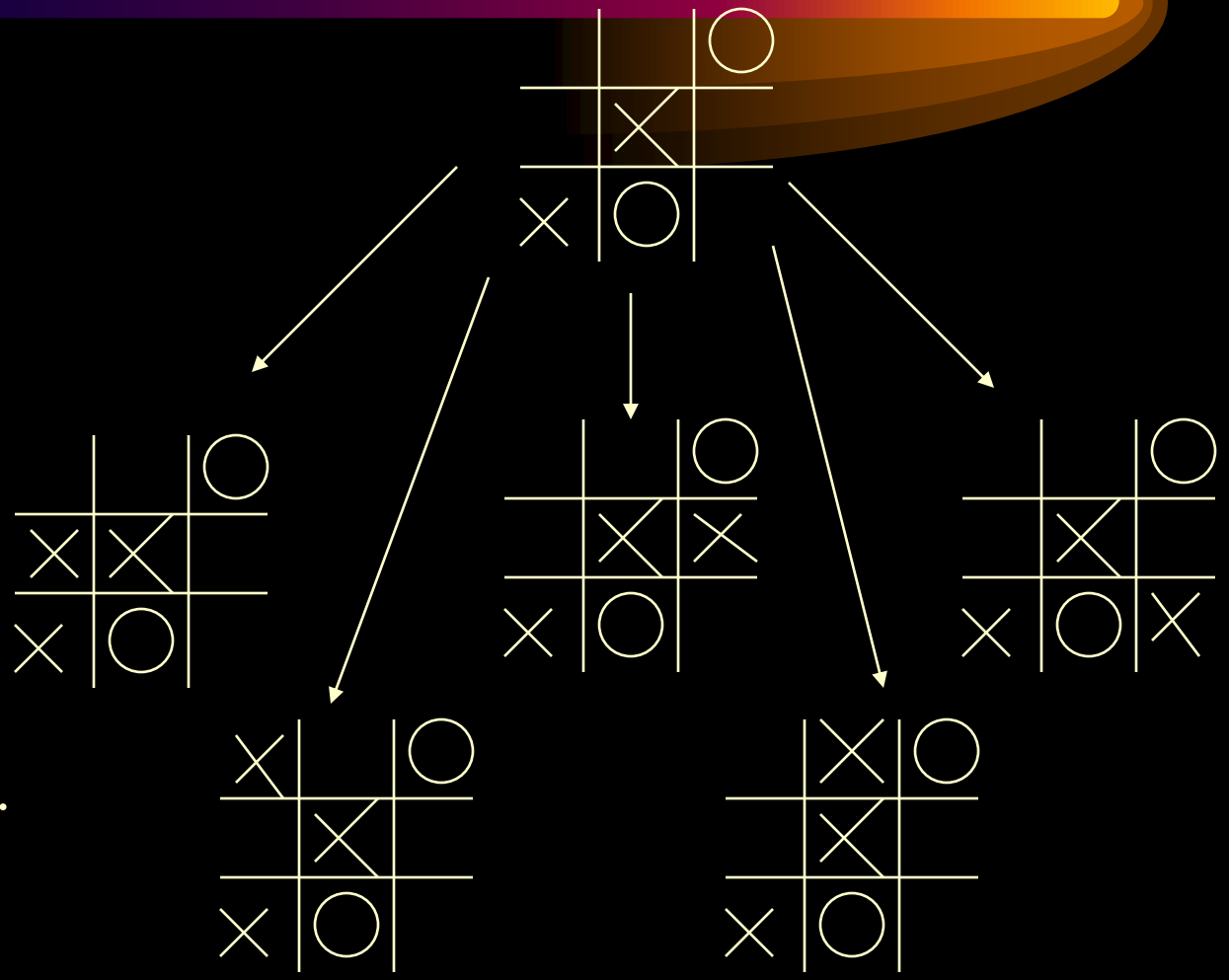
## Example 1:

Game problem:



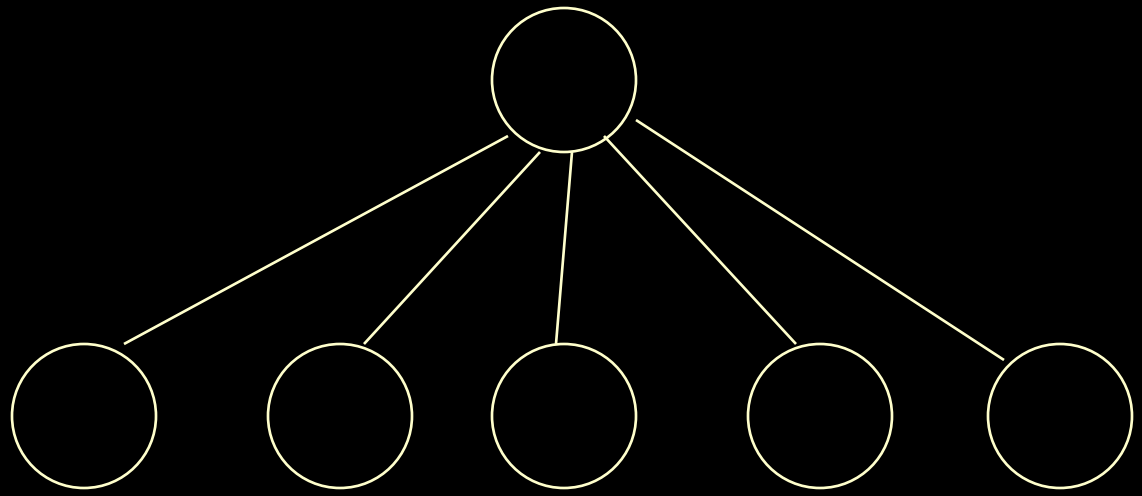
Next step:

x has five choices.



# The purpose and contents of the course

Example 1 uses a tree structure.



# The purpose and contents of the course

Example 2: Management of library catalogue by computer

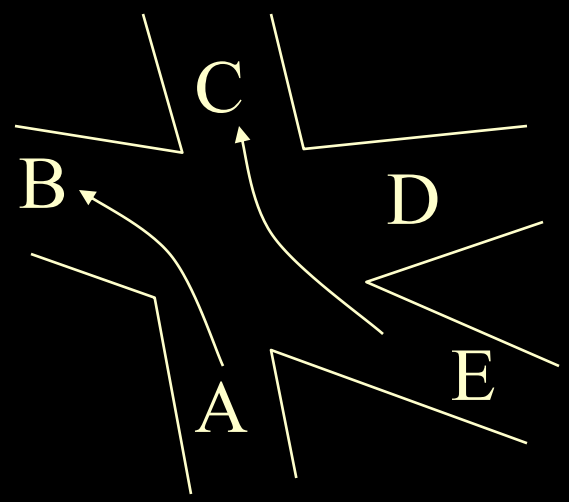
| 书名 | 作者名 | 登录号 | 分类 | 出版年月 |
|---|---|---|---|---|
| D.S. | Sartaj Sahni | 000001 | computer | 2000.1 |

It is a linear list.

# The purpose and contents of the course

## Example 3:

Management of the traffic lights in intersections




This is a graph

C, E are one-way road, there are 13 path to go.

Can go at the same time :

A→B  E→C

Cannot go at the same time:

E→B  A→D

# The purpose and contents of the course

2 . Prerequisite of other courses:

- Principles of compiling : use **stack** to compute expression and implement recursive procedure
- Operating System: use **queue** to implement job scheduling
- Database: use **B-tree**,**$B^+$ tree** to organize, store and load massive data in the hard memory.

...

# The purpose and contents of the course

3. Basic methods of algorithm analysis

standards of the performance of an algorithm:
time complexity, space complexity,
and accuracy

example: sorting

$a_1, a_2, a_3, \ldots\ldots, a_{n-1}, a_n$

$n-1+n-2+\ldots.+2+1= n(n-1)/2=(n^2-n)/2$

$O(n^2)$

$O(n*\log_2 n)$

4. Review Java and C++

# What is Data Structure

1. Data

   is the carrier of information.

   Data is a set of numbers , characters,and other symbols

- that can be used to describe the objective things.
- These symbols can be input into computers , identified and processed by the computer program.

# What is Data Structure

Data can divided into two classes:

- numerical data: int, float, complex,……
- non-numerical data: character, string, graph, voice…

# What is Data Structure

2．Data structure

A data structure is a data object together with the relationships among the data members that compose the object

Data_Structure={D,R}

D is a data object，

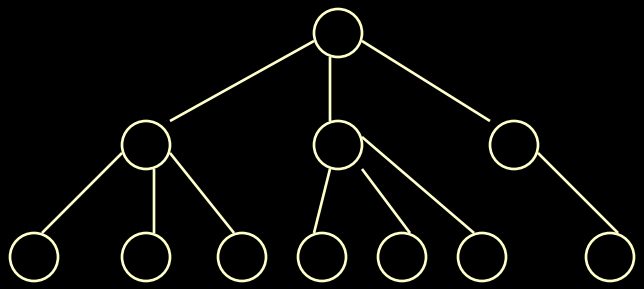
R is a limited set of relationships of all the data members in D.

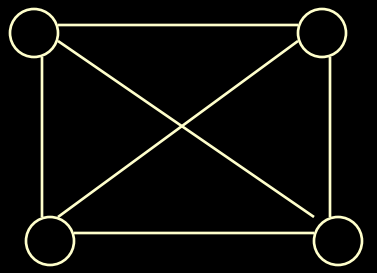
# What is Data Structure

Data structure

- Linear structure

- Non-linear structure





# What is Data Structurem

数据结构涉及三个方面：

数据的逻辑结构-----从用户视图看，是面向问题的。

数据的物理结构-----从具体实现视图看，是面向计算机的。

相关的操作及其实现。

Example:

学生表：逻辑结构-----线性表

物理结构-----数组, 拉链

操作-----插入, 删除, 查找

# ADT and OO

1. Data type

Definition: is a set of values together with a operation set that operate on these values.

Example:

int type

value set: {……,-2,-1,0,1,2,……}

operation set: {+, -, *, /, %, …}

# ADT and OO

Most of the programming languages provide a group of predefined data type.

- Atom data type — int, float, double……
- Structure data type—array, struct,……

# ADT and OO

## 2. ADTs: Abstract Data Types

是将类型和与这个类型有关的操作集合封装在一起的数据模型。

Abstract: is a method used to hide the information.

Example:

```
int x ;
   :
 x=735;
   :
```

abstract of int data type assignment

```
float x, y ;
   :
 x=x*y+3.0;
   :
```

abstract of float data type operation

# ADT and OO

Abstract data type:

is a new programming method that

part the usage and implementation, in order to encapsulate and hide the information.

思想：

将数据类型的使用与它的表示（机内存储）、实现（机内操作的实现）分开。更确切的说，把一个数据类型的表示及在这个类型上的操作实现封装到一个程序模块中，用户不必知道它。

**ADT** NaturalNunber **is**

- **objects:** 一个整数的有序子集合,它开始于0,结束于机器能表示的最大整数(MAXINT)。
- **Function:**
- Zero( ):NaturalNumber
- IsZero(x):Boolean
- Add(x,y):NaturalNumber
- Equal(x,y):Boolean
- Successor(x):NaturalNumber
- Subtract(x,y):NaturalNumber
- **end** NaturalNumber

# ADT and OO

3. OO:

object-oriented＝object＋class＋inherit＋communicate

object: attribute  values+ operates

example:  rectangle:  一个几何对象

attribute values : 左上角坐标, 右下角坐标,边线颜色, 内部颜色

operates : move(x,y);

setEdgeColor( c );

setInterColor( c ) ;

# ADT and OO

class: objects of same attributes and operates.

an instance is an object of the class.

different object has deferent attribute value

# ADT and OO

Inherit:

base class—integrate the same part (including attributes and operations) in all the derived classes

derived class—add the specific attributes and operations

Example: base class—polygon

derived class—quadrilateral,triangular

# ADT and OO

Communication: each class object communicate with others using messages.

Message: instructions that one class object used in order to require another class object to perform some operation.

# Algorithm definition

Algorithm : an operation sequence of soluting a problem

Characteristic: 1.finite

2. deterministic

3. initial action

4. sequence ends

# Algorithm definition

Program : is written by languages that can be performed by machine.

can't satisfy the finiteness.

For example, OS.

Algorithm: has multiple descriptive methods, such as language, graph, table.

# Algorithm definition

```
void selectsort(int a[ ],int n)
{   for (int i = 0; i <  n-1; i++)
     {    int k = i;
          for ( int j = i+1; j < n;  j++)
               if ( a[j] < a[k]) k = j;
          int temp = a[i]; a[i] = a[k]; a[k] = temp;
     }
}
```

# Mathematics Review

1. Exponents

$X^A X^B = X^{A+B}$

$X^A / X^B = X^{A-B}$

$(X^A)^B = X^{AB}$

$X^N + X^N = 2X^N$

$2^N + 2^N = 2^{N+1}$

# Mathematics Review

2. Logarithms( all logarithms are to the base 2)

DEFINITION :

$X^A = B$ if and only if $\log_X B = A$

THEOREM 1.1

$\log_A B = \log_C B / \log_C A$ ; A, B, C > 0, A != 1

THEOREM 1.2

$\log AB = \log A + \log B$; A, B > 0

3. Series

$$\sum_{i=0}^{N} 2^i = 1 + 2^1 + 2^2 + \ldots\ldots + 2^N = 2^{N+1} - 1$$

$$\sum_{i=1}^{N} i = 1 + 2 + 3 + \ldots\ldots + N = (N+1)*N/2$$

# Mathematics Review

## 4. Modular Arithmetic

We say that A is congruent to B modular N, written $A \equiv B \pmod{N}$, if N divides A-B.

Example : $81 \equiv 61 \equiv 1 \pmod{10}$

# Mathematics Review

## 5. The P Word (证明方法)

### 1). Proof by Induction

The first step is proving a base case.

the Next stap an inductive hypothesis is assumed.

theorem is assumed to be true for all cases up to some limit k. Using this assumption, the theorem is then shown to be true for the next value, which is typically k + 1. This proves the theorem( as long as k is finite).

Example: 例如 等比级数的和以及整数平方和的证明等.

# Mathematics Review

2). Proof by Contradiction

Proof by contradiction proceeds by assuming that the theorem is false and showing that this assumption implies that some known property is false, and hence the original assumption was erroneous.

Example: proof that there is an infinite number of primes

# A Brief Introduction to Recursion

1. Recursive

example:

define a function f, valid on nonnegative integers

$$f(x) = \begin{cases} 0 & x = 0 \\ 2f(x-1) + x^2 & x > 0 \end{cases}$$

f(1) = 1, f(2) = 6, f(3) =21, f(4) = 58

```
public static int f ( int x)
{   if ( x = = 0)  //base case
        return 0;
    else return 2*f(x-1) + x*x;  //recursive call
 }
```

# A Brief Introduction to Recursion

A nonterminating recursive method:

```
public static int bad( int n )
{   if ( n = = 0)
        return 0;
    else return bad( n/3 + 1) + n-1;
}
```

two fundamental rules of recursion:

1) Base cases.

2) Making progress.

# A Brief Introduction to Recursion

2. 1) direct recursive

   2) indirect recursive

Example 1 factorial function f(n)=n!

$$f(n)=\begin{cases} 1 & n<=1 \quad \text{(base)} \\ n*f(n-1) & n>1 \ \text{(recursive component)} \end{cases}$$

f(5)=5f(4)=5*4f( 3)=5*4*3f(2)=5*4*3*2f(1)=120

```
static long factorial (int n)
{   if ( n <= 1)
        return  1;
    else return n* factorial( n-1 )
 }
```

# A Brief Introduction to Recursion

```
public class ComputeFactorial
{   public static void main( String[ ] args )
    {   System.out. Println( “please enter a nonnegative  integer” );
         int n = MyInput.readInt( );

         Syatem.out.println( “Factorial of “ + n + “ is “ + factorial( n ) );
    }
    static long factorial (int n)
    {  if (n<=1) return 1;
       else  return n*factorial(n-1);
    }
}
```

# A Brief Introduction to Recursion

## Example 2: Compute Fibonacci number

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ……

$$\begin{cases} fib(0) = 0; \\ fib(1) = 1; \\ fib(n) = fib(n-2) + fib(n-1); \quad n>=2 \end{cases}$$

```
public static long fib(long n)
{   if ( ( n = = 0) || ( n = = 1 ) )
          return n;
    else
          return  fib(n-1) + fib(n-2);
}
```

Compute fib(4):

# A Brief Introduction to Recursion

```
public class ComputeFibonacci
{   public static void main( String args[ ])
    {   System.out.println( “Enter an index for the Fibonacci number “);
        int n = MyInput.readInt( );

        System.out.println( “Fibonacci number at index “ + n + “ is “ +
                                    fib(n));
    }
    public static long fib(long n)
    {   if (( n = = 0) || ( n = = 1))
            return 1;
        else
            return fib( n-1) + fib(n-2);
    }
}
```

# A Brief Introduction to Recursion

Example 3:

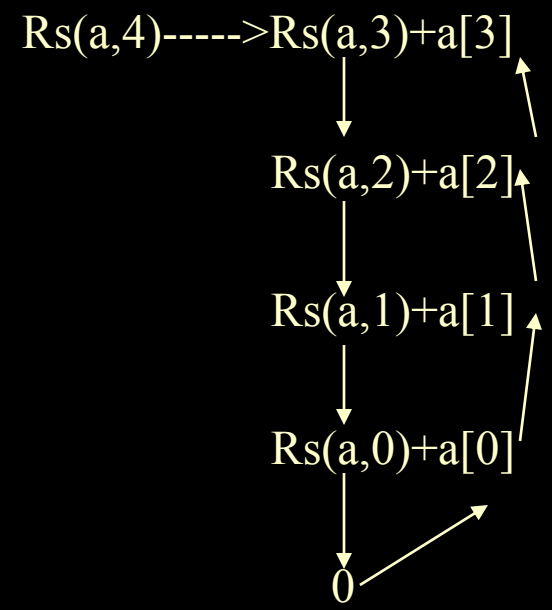
computes the sum of the elements a[0] through a[n-1]

a[0], a[1], …, a[n-2], a[n-1]

# A Brief Introduction to Recursion

```
public static  int  Rsum(int[] a , int n)
 { if (n>0)
       return Rsum(a,n-1)+a[n-1];
   return 0;
  }
```

$a_0$ $a_1$ $a_2$ $a_3$



# A Brief Introduction to Recursion

Example 4:

Permutation

{a,b,c} : abc, acb, bac, bca, cab, cba

permutation of n elements has n!

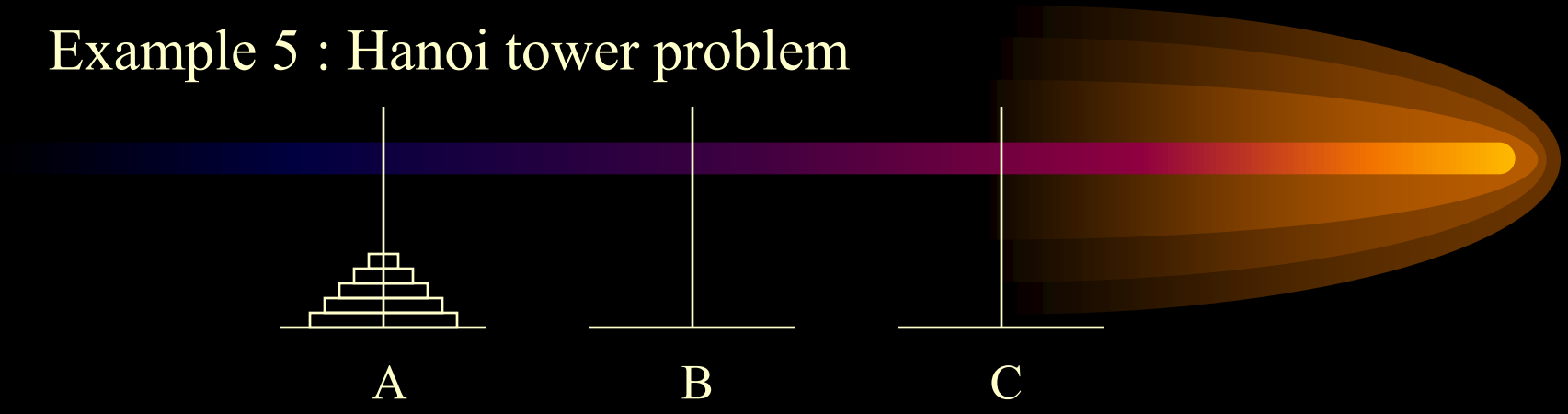
下面在黑板上来分析该问题：

# A Brief Introduction to Recursion

```
public static void perm(Char[ ] list ,  int k,  int m)
{    int i;
     if (k= =m) { for (i=0; i<=m; i++) cout << list[i];
                       cout << endl; }
     else{ for (i=k; i<=m; i++)
               { swap(list[k], list[i]);
                 perm(list, k+1, m);
                 swap(list[k], list[i]);
                }
          }
 }
```

# A Brief Introduction to Recursion

Example 5 : Hanoi tower problem




```
public void moveDISKs(int n, char fromTower, char
                                        toTower, char auxTower)
  if ( n= =1)
     Move disk 1 from the fromTower to the toTower;
  else
  {  moveDISKs(n-1, fromTower, auxTower, toTower);
     Move disk n from the fromTower to the toTower;
     moveDISKs(n-1, auxTower, toTower, fromTower);
   }
```

# A Brief Introduction to Recursion

```
public class TowersOfHanoi
{   public static void main(String[ ] args)
    {   System.out.printLn( “Enter number of disks” );
        int n = MyInput.readInt( );

        System.out.printLn( “The move are:”);
        moveDISKs(n, ‘A’, ‘B’, ‘C’);
     }
    public static void moveDISKs(int n, char fromTower, char
                                        toTower, char auxTower)
   {  if ( n= =1)
        System.out.printLn( “move disk “ + n + “from “+
                                fromTower +” to” + toTower);
     else
     {   moveDISKs(n-1, fromTower, auxTower, toTower);
        System.out.printLn( “Move disk “ + n + “from “ +
                                 fromTower + “ to “ + toTower );
        moveDISKs(n-1, auxTower, toTower, fromTower);
     }
    } }
```

# Generic Objects in Java

Throughout this text, we will describe algorithms and data structures that are type independent.

- IntCell —— nongeneric class
- MomoryCell —— generic class

# Generic Objects in Java

1. IntCell class

```
public class IntCell
{   public IntCell ( ){ this ( 0 ); }
    public IntCell( int initialValue )
       {   storedValue = initialValue ; }

    public int read ( ) { return storedValue; }
    public void write ( int x ) {  storedValue = x; }

    private int storedValue;
}
```

# Generic Objects in Java

1) public and private

memble of public : may be accessed by any method in any class .

member of private : may only be accessed by methods in its class.

specifier is omitted (package-friendly visibility) :

a data member whose visibility specifier is omitted is visible to other classes in the same package.

# Generic Objects in Java

## 2) constructor

A class may define methods that describe how an instance of the class is constructed; these are called constructors.

If no constructor is explicitly defined, one that initializes the fields using language defaults is automatically generated.

# Generic Objects in Java

## 3) This

- In many cases,the zero-parameter constructor can be implemented by calling anather constructor.

  By using **this** , we avoid replicating code logic in separate constructors.

- A different use of **this** is as a reference to the object being acted upon.

# Generic Objects in Java

```
public class TestIntCell
{   public static void main ( String [ ] args )
    {   IntCell m = new IntCell ( );
        m. write( 5 );
        System.out.println( “Cell contents: “ + m.read( ) );
    }
}
```

# Generic Objects in Java

1) static and main

The **main** method must be **static** , meanimg that it applies to the **TestIntCell** class instead of a single instance of the class.

Each class may declare a **main** method that will be used when the java interpreter is called for that class.

复习一下static

2) Object creation.

Objects are created by using **new**.

# Generic Objects in Java

3) Method calls.

m.write(5);

m.read( )

m.storedValue //be illegal

# Generic Objects in Java

2. MemoryCell class

design a class that works for any type of Object.

example: sorting of array $a_0, a_1, a_2, a_3, \ldots\ldots a_{n-1}$

- C++: use templete

program1.1:

```
int Abc(int a,int b,int c)
{return a+b+b*c+(a+b-c)/(a+b)+4;     }
```

program1.2:

```
float Abc(float a,float b,float c)
{return a+b+b*c+(a+b-c)/(a+b)+4;     }
```

# Generic Objects in Java

program1.1 and 1.2 differ only in the data type of the formal parameters and of the value returned.

We can write a generic code in which the data type is a variable whose value is determined by the compiler.This generic code is written using the template statement in program1.3

# Generic Objects in Java

## Program1.3

```
template<class T>
 T Abc(T a,T b,T c)
 {return a+b+b*c+(a+b-c)/(a+b)+4;  }
```

From this generic code the compiler can construct 1.1 by substituting int for T and 1.2 by substituting float for T.

# Generic Objects in Java

- Java:

1) use inheritance(pre-java 5)

all objects are subclasses of Object.

```
public class MemoryCell
{  public Object read( )
     {  return storedValue; }
    public void write( Object x )
     {  storedValue = x; }

    private  Object storedValue;
}
```

1.5 A generic MemoryCell class (pre-java 5)

# Generic Objects in Java

Two problem:

First problem: We must downcast to the correct type

```
public class TestMemoryCell
{   public static void main( String [ ] args )
    {   MemoryCell m = new MemoryCell (  ) ;

        m.write( "37 " );
        string  val = ( String) m.read( );
        System.out.println( "Contents are : " + val )
    }
}
```

Using the generic MemoryCell class (pre-Jave 5)

the read method for MemoryCell returns an Object.

# Generic Objects in Java

Second problem Although all objects are subclasses of Object, the primitive types boolean, short, char, byte, int, long, float, double are not objects. Thus we cannot directly use MemoryCell to store a primitive value.

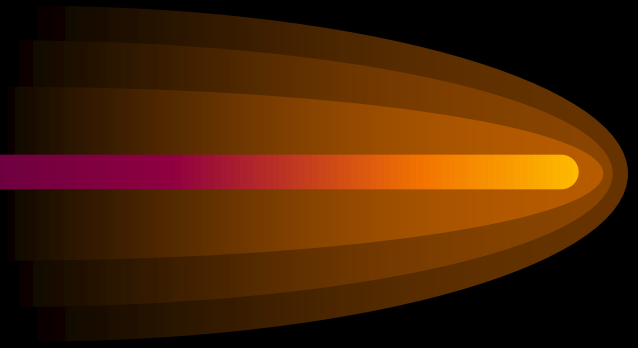
We must use wrapper class provide by Jave. Wrapper class store a single primitive value and can be used wherever an Object is need.

the wrapper class provides a method that can be used to access the primitive value it stores.

two example:

a) For the Integer wrapper, this method is named intValue.

# Generic Objects in Java



```
public class Integer
{    public   Integer( int x)
     {    value = x; }
      public  int intValue( )
      {   return value; }

      private int value;
}
```

# Generic Objects in Java

```
public class wrapperdemo
{   public static void main ( String [ ] args )
    {       MemoryCell m = new Memorycell (  ) ;

            m.write ( new Integer ( 37 ) ) ;
            Integer wrapperVal = ( Integer ) m . Read ( ) ;
            int val = wrapperVal . intValue (  ) ;
            System . Out . Println ( " Contents are : " + val ) ;
     }
}
```

1.7 An illustration of Integer wrapper class

To get the int value that is hidden inside the Integer object, we must convert the result of read back to an Integer , and then use mathod intValue access the value . This involves using a type conversion.

# Generic Objects in Java

2) Java 5 supports generic classes that are very easy to use

```
public class GenericMemoryCell <Any Type >
{    public AnyType read( )
     {    return storedValue; }
     public void write( AnyType x )
      {    storedvalue = x; }
     private AnyType storedvalue ;
 }
```

1.9 Generic implementation of the MemoryCell class

# Generic Objects in Java

- When a generic class is specified, the class declaration includes one or more type parameters after the class name
- User can create types such as GenericMemoryCell<String>
  GenericMemoryCell<Integer>
  but can not create GenericMemoryCell<int>
- Inside the GenericMemoryCell class declaration,
  we can declare fields of the generic type and
  methods that use the generic type as a parameter or return type

# Generic Objects in Java

- Interfaces can also be declared as generic.

  prior to Java 5, the Comparable interface was not generic.

  In Java 5, the Comparable class is generic.

```
package java.lang;
public interface Comparable<AnyType>
{     public int compareTo (  AnyType other ) ;
}
```

1.10 Comparable interface, Java 5 version which is generic

# Generic Objects in Java

```
class BoxingDemo
{   public static void main ( String [ ] args )
    {    GenericMemoryCell<Integer> m =
                        new GenericMemoryCell<Integer> ( ) ;
                                insert a call to the Integer constructor behind the scenes
         m . Write ( 37 ) ;       insert a call to the intValue method behind the
         int val = m . Read ( ) ;                                          scenes

         System . Out . Println ( " contents are : " + val ) ;
    }
}
```

1.11 Autoboxing and unboxing

**编译程序在箭头处分别调用Integer构造函数与intValue方法

# Generic Objects in Java

b) Implementing Generic findMax

The MemoryCell class required no special properties of Object ; the only operations performed were reference assignments, which are always available.

Finding the maximum item in an array of items does require a special property; we must be able to compare two items and decide which is larger.

# Generic Objects in Java

```
Comparable findMax ( Comparable [ ] a )
{     Comparable maxValue = a[ 0 ];
      for ( int i = 1; i < a.length; i++ )
         if ( maxValue.lessThan ( a[i] ) )
              maxValue = a[i];

       return maxValue;
}
```

a generic findMax algorithm.

Notice that the objects that are manipulated are not Object, But instead are Comparable.

# Generic Objects in Java

Comparable is an interface.

In java, an interface declares a set of methods that must be implemented.

- In this example, any class that is Comparable must provide an implementation of lessThan
- A class that is Comparable must also declare so by using the implements clause.

```
public interface Comparable
{   int lessThan( Comparable rhs );
}
```

compareTo

# Generic Objects in Java

```
Comparable findMax ( Comparable [ ] a )
 {     Comparable maxValue = a[ 0 ];
       for ( int i = 1; i < a.length; i++ )
          if ( maxValue.compareTo(a[i])<0 )
                maxValue = a[i];

        return maxValue;
 }
```

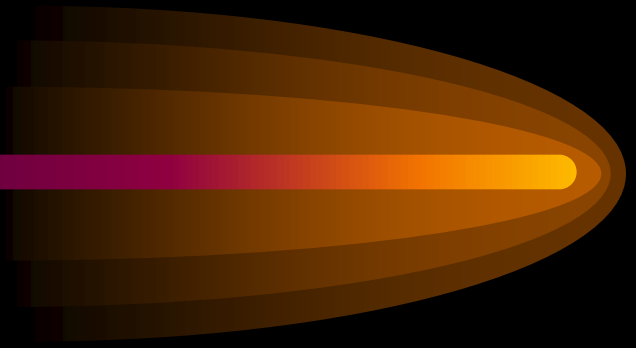
a generic findMax algorithm.

# Generic Objects in Java

In the book：

```
Class FindMaxDemo
{    public static comparable findMax(   Comparable [ ] arr )
     {     int maxIndex = 0;
           for ( int i = 1; i < arr.length; i++ )
                if ( arr[ i ] . compareTo( arr[ maxIndex ] ) > 0 )
                          maxIndex = i ;
            return arr[ maxIndex ];
      }
      public static void main ( String [ ] args )
      {     Shape [ ] sh1 = {  new  Circle( 2.0 ),  new   Square( 3.0 ),
                                    new Rectangle( 3.0, 4.0 ) }
            String [ ] sta = { “Joe”, “ Bob”, “Bill”, “Zeke” };
            System.out.println(  findMax( sh1 )  );
            System.out.println(  findMax( st1 ) );
      }
}
```

# Generic Objects in Java



```
//another Example:
public interface Comparable
{   public int compareTo(object o );
}

public class Max
{   public static Comparable max( Comparable o1, Comparable o2 )
     {    if ( o1. compareTo( o2 ) > 0 )
               return o1;
          else  return o2;
     }
}
```

# Generic Objects in Java

```
class Circle
{   private double radius;
    public Circle( ) {    radius = 1.0; }
    public Circle( double r ) {   radius = r ; }
    public double getRadius ( ) {   return  radius; }
    public void set Radius( double newRadius )
        {  radius = newRadius; }
    public double findArea( )
        {  return  radius * radius*3.14159; }
}
```

# Generic Objects in Java

```
class ComparableCircle extends Circle implements Comparable
{   public ComparableCircle ( double r )
    {    super ( r );
    }
    public int compareTo (object o )
    {    if ( getRadius( ) > ( ( Circle ) o ) . getRadius ( ) )
             return 1;
         else if ( getRadius ( ) < ( ( Circle ) o ) . getRadius ( ) )
                  return –1;
              else  return 0;
    }
}
```

# Generic Objects in Java

```
public class TestInterface
{   public static void main( String[ ] args )
    {   ComparableCircle circle1 = new ComparableCircle( 5 );
        ComparableCircle circle2 = new ComparableCircle( 4 );
        Comparable circle = Max . max (circle1, circle2 );
        System.out.println ( " The max circle's radius is " +
                              ( ( Circle ) circle ) . getRadius ( ) );
        System.out.println ( circle );
    }
} //another Example
```

# Exceptions

Java的异常处理提供对运行时错误的语言级处理机制。

1. 三类error

语法error，语义error, 逻辑 error

1) 语法error

通常在编译时发现，又称编译错。

如：标识符未声明，类型不匹配，缺少分号等。

2) 语义error

只有到程序运行时才发现，又称运行错。

如：除数为0, 数组下标越界等。

打开的文件不存在，网络连接中断等。

3) 逻辑 error

运行结果与期望值不符。这类错误最难确定与排除。

# Exceptions

2. 运行时的error又分为两类 (根据错误的性质)

1) 错误(error)------运行时遇到硬件或OS错误。

如，内存溢出(out of Memory);
虚拟机错误。

} 不可恢复的异常

2) 异常(exception)------硬件与OS都正常，程序遇到的运行错。

如，除数为0, 网络连接中断，打开文件发现文件不存在等。

3. Java的异常处理

1) 异常处理语句

# Exceptions

```
try
{
    语句1                    //存在潜在异常的代码
}
catch(异常类  异常对象)
{
    语句2                    //捕获到异常并进行处理的代码
}
finally
{
    语句3                    //最后必须执行的代码，无论是否捕获到异常
}
```

# Exceptions

```
public class Try2
{   public static void main( string args[ ])
    {   int i = 0;     int a[ ] = { 5,6,7,8};
        for( i = 0; i < 5; i++)
        {   try
            {   system.out.print("a[" + i + "]/" + i + " = " + (a[i]/i));
            }
            catch(ArrayIndexOutOfBoundsException e)
            {   system.out.print("捕获数组下标越界异常！ ");    }
            catch( ArithmeticException e)
            {   system.out.print("捕获算术异常！ ") }
            catch(Exception e)
            {   system.out.print("捕获" + e.getMessage( ) + "异常!" );
                                        //显示异常信息                    }
            finally
            {   system.out.println(" i = " +i );    }
        } //for
        system.out.println( " 继续！ " ） ;
    }
}
```

# Exceptions

运行结果为：

捕获算术异常！ i = 0

a[1]/1 = 6 i = 1

a[2]/2 = 3 i = 2

a[3]/3 = 2 i = 3

捕获数组下标越界异常！ i = 4

继续！

# Exceptions

2) 抛出异常

- 抛出自定义异常对象

  throw 异常对象

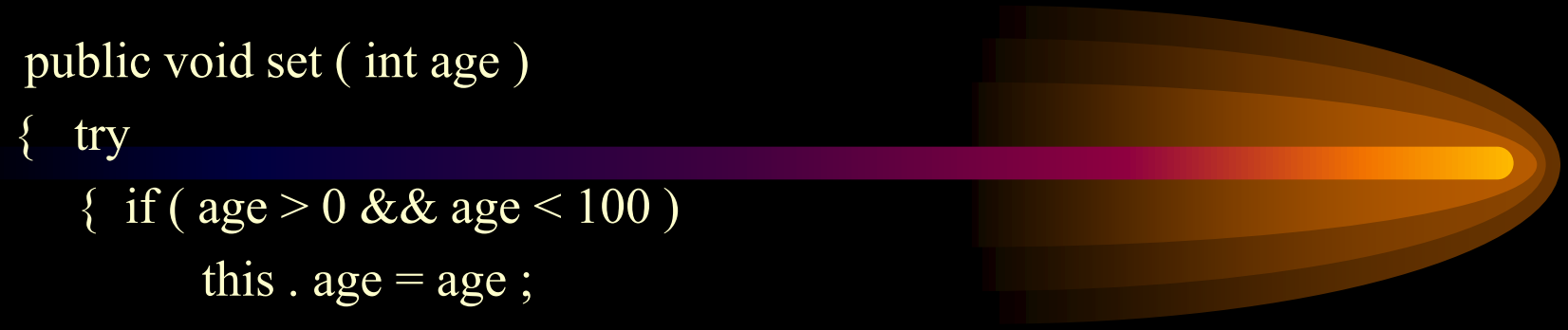
由throw语句抛出的异常也必须由try语句捕获并处理。

example

```
public void set(int age)
{   if(age > 0 && age < 100 )
        this.age = age ;
    else
        throw new Exception ( " IllegaAgeData" ) ;  //抛出异常
}
```

抛出异常和处理可以在一个方法中，也可分别在不同的方法中。一般而言，一个方法通过throw抛出异常，由方法的调用者捕获并处理该异常对象。

# Exceptions



```
public void set ( int age )
{   try
    {  if ( age > 0 && age < 100 )
            this . age = age ;
       else  throw new Exception ( " IllegalAgeData " ) ;
    }
    catch ( Exception e )
    {     system . Out . Println ( e . toString ( ) ) ;
    }
}
```

# Exceptions

- 方法声明抛出异常的throws子句

example

```
public void removeAny( ) throws Underflow
{    if ( isEmpty( ) )
         throw new Underflow( );
      …….
}
```

# Input and Output

import java.io.*

1. Basic Stream Operations

System.in standard input;

System.out standard output

System.err standard error

output in java :is done almost exclusively by String concatenation, with no built-in formatting.

use toString

input in java : reading formatted input

is to read a single line into a String object using readLine. The readLine method reads until it encounters a line terminator or end of file.

To use readLine, we must first construct a BufferedReader object from an InputStreamReader object that itself constructed from System.in.

# Input and Output

```
import java.io.*;
public class DivideByTwo
{   public static void main( String [ ] args )
    {   bufferedReader in = new BufferedReader( new
                                    InputStreamReader ( System.in ) );
        int x;
        String oneLine;
        System.out.println( "Enter an integer:" );
        try
        {  oneLine = in.readLine( );   //  IOException
           x = Integer.parseInt( oneLine );  //NumberFormatException
           System.out.println( "Half of x is " + ( x/2) );
         }
         catch( Exception e )
              {  system.out.println( e ); }
      }
}
```

# Input and Output

Recall that to read a single primitive type, such as an int:

1) use readLine to read the line as a String.

2) then apply a method to generate the primitive type from the String. For int , we can use parseInt.

## 2. The StringTokenizer Object

Sometimes we have several items on a line. For instance, suppose each line has two **int**s .

Java provides the **StringTokenizer** object to separate a **String** into **tokens.**

# Input and Output

```
import java.io.*
import java.util.*;
public class MaxTest
{   public static void main( String [ ] args )
    {   BufferedReader in =  new BufferedReader
                    ( new InputStreamReader( System.in ) );
        String oneLine;
        StringTokenizer str;
        int x;   int y;
        System.out.println( "Enter 2 ints on one line: " );
        try
        {    oneLine = in.readLine( );
             str = new StringTokenizer( oneLine );
             if( str . countTokens( ) != 2 )
                  throw new NumberFormatException( );
             x = Integer . parseInt( str . nextToken( ) );
             y = Integer . parseInt( str . nextToken( ) );
             System.out.println( "Max: " + Math.max( x, y ) ) ;
        }
        catch ( Exception e )
           {     System.err.println( "Error: need two ints" ); }    }   }
```

# Input and Output

- To use **StringTokenizer** , provide the **Import** directive

  import java.util.*;

- By default, tokens are separated by white space.

example:

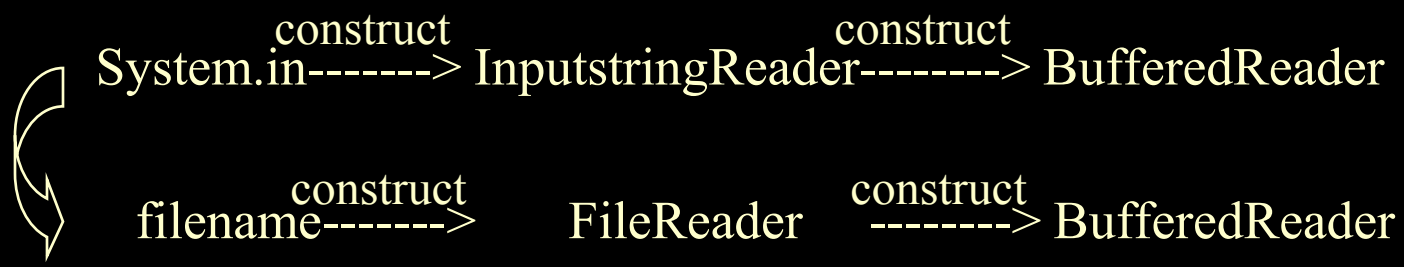
" I am learning Java now "

# Input and Output

3. Sequential Files

One of the basic rules of java is that what works for terminal I/O as works for files.

To deal with a file:



# Input and Output

```
import java.io.*;
public class ListFiles
{    public static void main( String [ ] args )
     {    if( args.length = = 0 )
               System.out.println( “No files specified” );
          for( int i = 0; i < args.length; i++ )
               listFile( args[ i ] );
     }
     public static void listFile( String fileName )
     {     FileReader theFile;
          BufferedReader fileIn = null;
          String oneLine;

           Systnm.out.println( “FILE: “ + fileName );
```

# Input and Output

```
        try
        {    theFile = new FileReader( fileName );
             fileIn = new BufferedReader( theFile );
             while( ( oneLine = fineIn.readLine( ) ) != null )
                    System.out.println( oneLine );
        }
        catch(  exception e )
            {   System.out.println( e ); }

         // close the stream
        try
        {     if( fileIn != null )
                    fileIn.close( );
        }
        catch( IOException e ) { }
    }
}
```

# Code Organization

1. Packages

   package DataStructures;

   Most of our classes will be found in a package named DataStructures. This includes the Overflow and Underflow exception classes.

# Chapter 1

Exercises:

1. Write a recursive method that returns the number of 1's in the binary representation of N. Use the fact that is equal to the number of 1's in the representation of N/2, plus 1, if N is odd.
2. Write the routines wise the following declarations:

```
public void permute( String str );
private void permute( char [ ] str, int low, int high )
```

The first routine is a driver that calls the second and prints all the permutations of the characters in String str. If str is "abc", then the strings that are output are abc, acb, bac, bca, cab,and cba. Use recursion for the second routine.

3. 已知a[n]为整型数组，试写出实现下列运算的递归算法。
   1）求数组a中的最大整数。
   2）求n个整数的平均值。

# Chapter 1

4. Write a recursive method that calculates and returns the length of a linked list.
5. Check recursively if the following objects are palindromes:
   a. A word
   b. a sentence ( ignoring blanks, lower- and uppercase differences, and punctuation marks so that “Madam, I’m Adam” is accepted as a palindrome)

# Chapter 1

实习题:

1. 找出从自然数 1, 2, …, n 中任取r个数的所有组合, 编一个递归算法.

   例子: n = 5　　1 2 3 4 5

   　　r = 3　　5 4 3

   　　5 4 2

   　　5 4 1

   　　5 3 2

   　　5 3 1

   　　5 2 1

   　　4 3 2

   　　4 3 1

   　　4 2 1

   　　3 2 1

2. 实现Hanoi塔