CS202 Java Object Oriented Programming

Review of Language Basics

Chengyu Sun
University of California, Santa Barbara

Overview

- ◈Programming environments
- ◈Basic program structure
- ◈Variables and types
- ◈Operators
- ◈Methods and recursion
- ◈Arrays

JDK

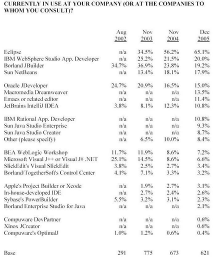
- ◈http://java.sun.com
- ◈`javac` Welcome.java
- ◈`java` Welcome

Java IDEs

- ◈ JBuilder
  - Commercial with a scaled-down free version (`JBuilder Foundation`)
- ◈ Eclipse
  - http://www.eclipse.org
  - High quality and free
  - Many tools and plug-ins have to be installed separately
- ◈ Netbeans
  - http://www.netbeans.org
  - All-in-one package
  - Slow on older computers

IDE Usage Statistics

- ◈Survey by *BZ Research* in December 2005
  - Eclipse: 65.1%
  - JBuilder: 19.2%
  - Netbeans: 17.9%



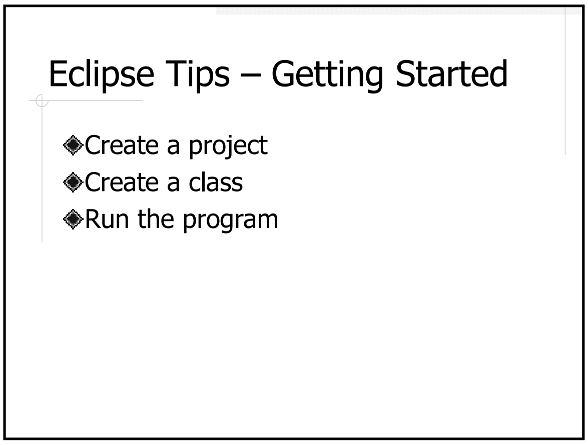
Java Use and Awareness Study
© BZ Research December 2005

QUESTION 8. WHICH JAVA DEVELOPMENT ENVIRONMENTS ARE CURRENTLY IN USE AT YOUR COMPANY (OR AT THE COMPANIES TO WHOM YOU CONSULT)?

	Aug 2002	Nov 2003	Nov 2004	Dec 2005
Eclipse	n/a	34.5%	56.2%	65.1%
IBM WebSphere Studio App. Developer	n/a	25.2%	21.5%	20.0%
Borland JBuilder	34.7%	36.9%	23.8%	19.2%
Sun NetBeans	n/a	13.4%	18.1%	17.9%
Oracle JDeveloper	24.7%	20.9%	16.5%	15.0%
Macromedia Dreamweaver	n/a	n/a	n/a	13.5%
Emacs or related editor	n/a	n/a	n/a	11.4%
JetBrains IntelliJ IDEA	3.8%	8.1%	12.3%	10.8%
IBM Rational App. Developer	n/a	n/a	n/a	10.8%
Sun Java Studio Enterprise	n/a	n/a	n/a	9.3%
Sun Java Studio Creator	n/a	n/a	n/a	8.7%
Other (please specify)	n/a	6.5%	10.0%	8.4%
BEA WebLogic Workshop	11.7%	11.9%	8.6%	7.2%
Microsoft Visual J++ or Visual J# .NET	25.1%	14.5%	8.6%	6.6%
SlickEdit's Visual SlickEdit	3.8%	2.5%	2.7%	3.4%
Borland/TogetherSoft's Control Center	4.1%	7.1%	3.3%	3.2%
Apple's Project Builder or Xcode	n/a	1.9%	2.7%	3.1%
In-house-developed IDE	n/a	2.7%	2.4%	2.6%
Sybase's PowerBuilder	5.5%	3.2%	3.1%	2.3%
Borland Enterprise Studio for Java	n/a	n/a	n/a	2.1%
Compuware DevPartner	n/a	n/a	n/a	0.6%
Xinox JCreator	n/a	n/a	n/a	0.6%
Compuware's OptimalJ	1.0%	1.2%	0.6%	0.4%
Base	291	775	673	621

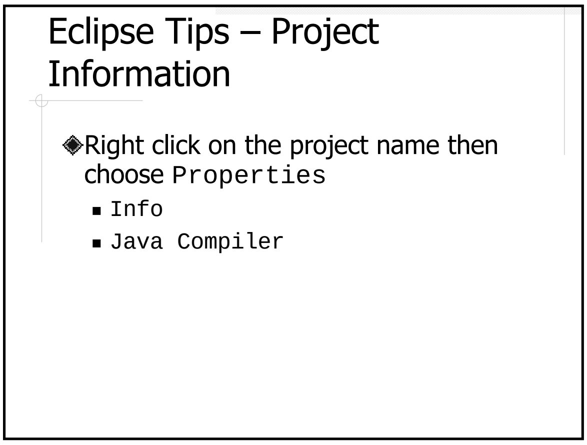
Eclipse Tips – Views and Perspective

- ◈Each block is called a View
  - `Window -> Show View`
  - `Package Explorer` view
- ◈A number of views constitute a Perspective
  - `Window -> Open Perspective`
  - `Java` perspective



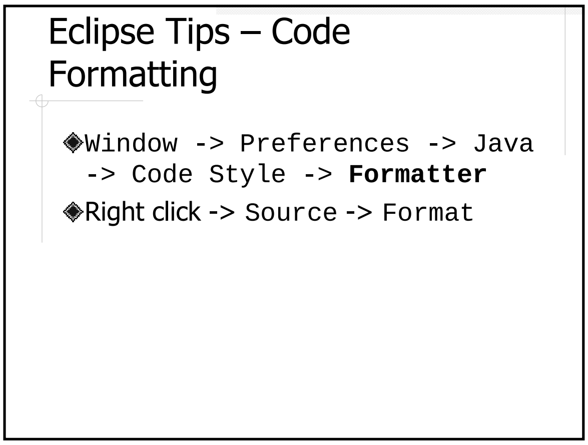
Eclipse Tips – Getting Started

- Create a project
- Create a class
- Run the program



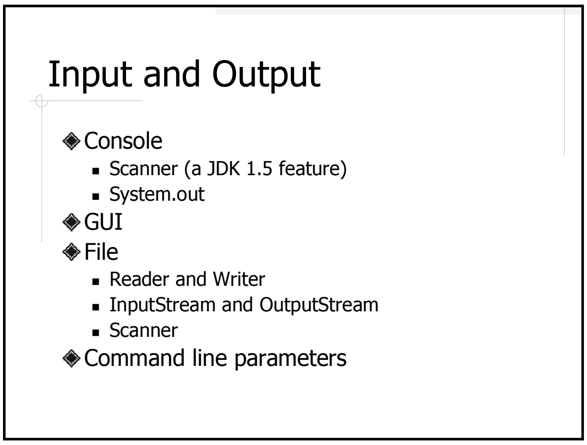
Eclipse Tips – Project Information

- Right click on the project name then choose `Properties`
  - `Info`
  - `Java Compiler`



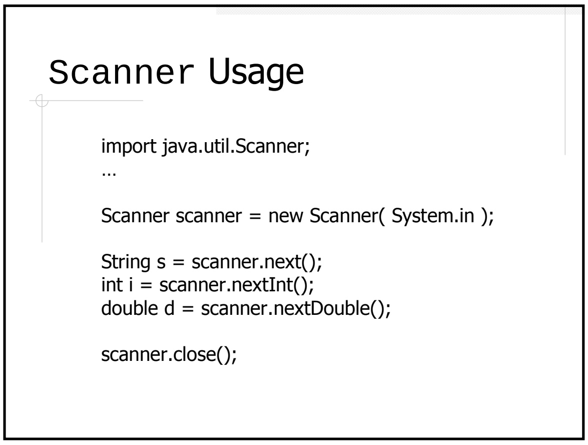
Eclipse Tips – Code Formatting

- `Window -> Preferences -> Java -> Code Style ->` **`Formatter`**
- Right click -> `Source -> Format`



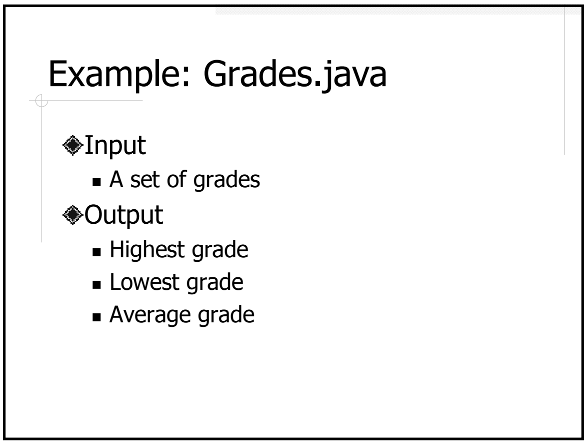
Input and Output

- Console
  - Scanner (a JDK 1.5 feature)
  - System.out
- GUI
- File
  - Reader and Writer
  - InputStream and OutputStream
  - Scanner
- Command line parameters



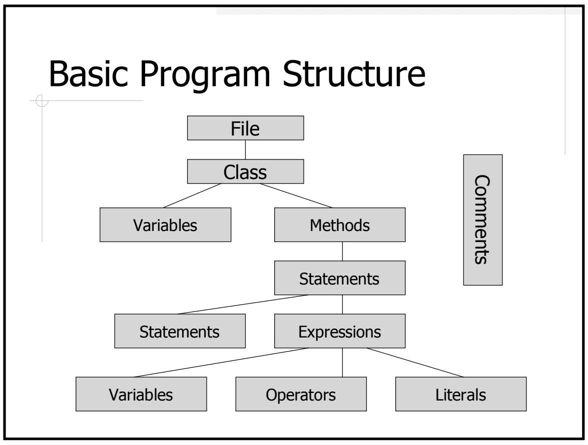
`Scanner` Usage

```
import java.util.Scanner;
…

Scanner scanner = new Scanner( System.in );

String s = scanner.next();
int i = scanner.nextInt();
double d = scanner.nextDouble();

scanner.close();
```



Example: Grades.java

- Input
  - A set of grades
- Output
  - Highest grade
  - Lowest grade
  - Average grade



Basic Program Structure

File

Class

Variables

Methods

Statements

Statements

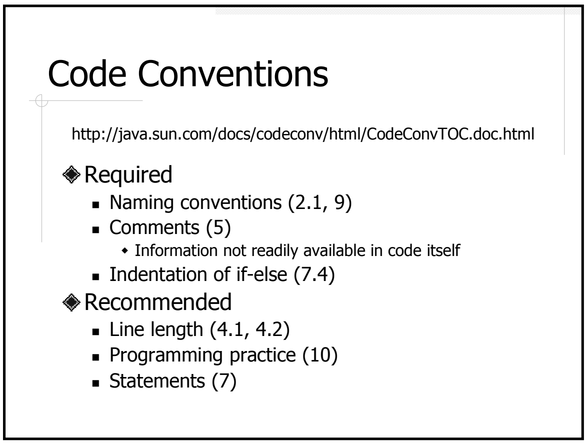
Expressions

Variables

Operators

Literals

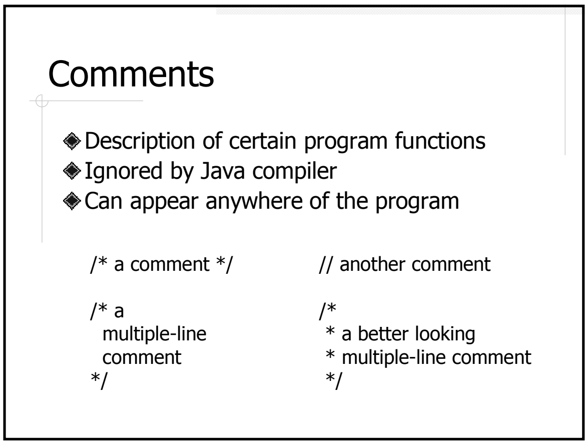
Comments



Code Conventions

http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

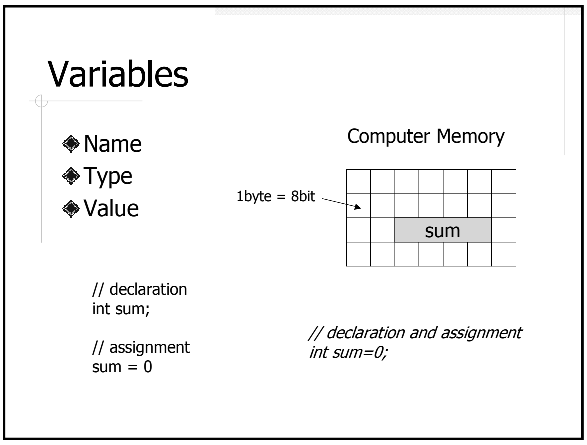
- Required
  - Naming conventions (2.1, 9)
  - Comments (5)
    - Information not readily available in code itself
  - Indentation of if-else (7.4)
- Recommended
  - Line length (4.1, 4.2)
  - Programming practice (10)
  - Statements (7)



Comments

- Description of certain program functions
- Ignored by Java compiler
- Can appear anywhere of the program

```
/* a comment */              // another comment

/* a                         /*
  multiple-line               * a better looking
  comment                     * multiple-line comment
*/                            */
```



Variables

- Name
- Type
- Value

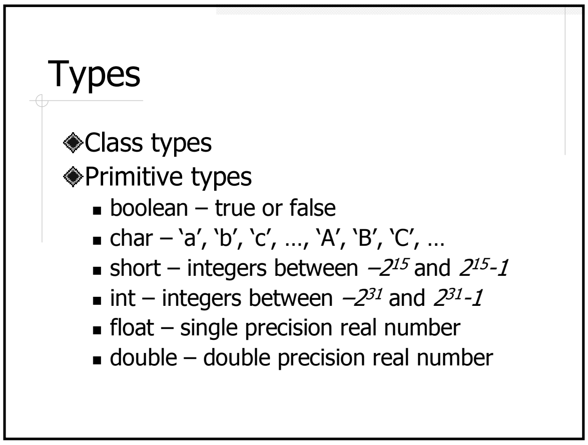
Computer Memory

1byte = 8bit

sum

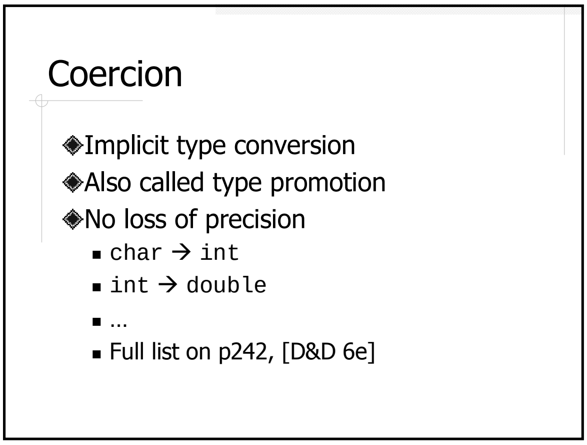
```
// declaration
int sum;

// assignment
sum = 0
```

// declaration and assignment
int sum=0;



Types

- Class types
- Primitive types
  - boolean – true or false
  - char – ‘a’, ‘b’, ‘c’, …, ‘A’, ‘B’, ‘C’, …
  - short – integers between -2^{15} and $2^{15}-1$
  - int – integers between -2^{31} and $2^{31}-1$
  - float – single precision real number
  - double – double precision real number



Coercion

- Implicit type conversion
- Also called type promotion
- No loss of precision
  - `char` → `int`
  - `int` → `double`
  - …
  - Full list on p242, [D&D 6e]

Cast

- ◈Explicit type conversion
- ◈*Possible* loss of precision
- ◈Syntax: (*Type*) *Expression*

```
double    number = 3.6;

int    integer_part = (int) number; // cast
double    fraction_part = number – integer_part;
```

Values (a.k.a. Literals)

- ◈boolean: true, false
- ◈char: 'a', 'b', … , 'A', 'B', …, '1', '2', …
- ◈float, double: -0.1, 99.99, 1.1e13
- ◈short, int: -10, 203, 0x11, 011 …

Number Systems

- ◈Base-2 (Binary)
  - 0, 1
- ◈Base-8 (Octal)
  - 0, 1, … , 7
- ◈Base-10 (Decimal)
  - 0, 1, … , 9
- ◈Base-16 (Hexadecimal)
  - 0, …, 9, A, B, C, D, E

1	1	0	1
2^3	2^2	2^1	2^0

Bin: $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

1	1	0	1
16^3	16^2	16^1	16^0

Hex: $1 \times 16^3 + 1 \times 16^2 + 0 \times 16^1 + 1 \times 16^0$

Operators

- ◈Arithmetic
  - +
  - -
  - *
  - /
  - %
- ◈Assignment
  - =
  - +=, -=, *=, /=, %=
- ◈Increment and decrement
  - ++
  - --

More Operators

- ◈Relational
  - ==, !=
  - >, <
  - >=, <=
- ◈Conditional
  - ?:
- ◈Logical
  - Negation: !
  - AND: &&
  - OR: ||

Precedence

- ◈ Determines the evaluation order of different types of operators
- ◈ Or, *parenthesis* to the rescue
- ◈ Exercise: check out the operator precedence table in the textbook (Appendix A)

↑ Increment/decrement
Arithmetic
Logical
Assignment ↑

```
a + b * c – d
a + b * c – d++
a + b * c - ++d

!a && b || c && d && a > d
!a && (b || c) && d && (a > d)
```

Associativity

- ◈Determine the evaluation order of the operators with the same precedence
- ◈Left-associative
  - Most operators are left-associate
  - E.g. *a + b + c*
- ◈Right-associative
  - E.g. ??

Control Statements

- ◈Branch
  - if
  - if … else
- ◈Switch
  - switch
- ◈Loop
  - while
  - do … while
  - for
- ◈Break and continue
  - break
  - continue

Method

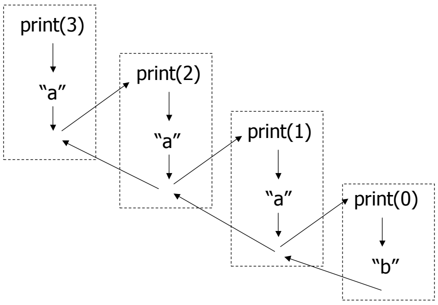
- ◈Header
  - Access modifier
  - Return type
  - Name
  - Parameter list
- ◈Body

Recursion

- ◈A method calls itself

```
void print( int n )
{
   if( n <= 0 ) System.out.println();
   else
   {
      System.out.print("a");
      print(n-1);
   }
}
```

Recursive Process



Ending Condition

- ◈When the recursion should stop
- ◈To avoid infinite recursion, make sure the ending condition
  - Exists
  - Reachable
  - Comes before the recursive call

Simple Recursion Examples

- Factorial
  - f(n) = 1*2*3*…*n
- Fibonacci series
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, …
  - Definition
    - fibonacci(0) = 0
    - fibonacci(1) = 1
    - fibonacci(n) = fibonacci(n-1)+fibonacci(n-2)

When Can We Use Recursion?

- A problem itself is recursively defined
  - Fibonacci → f(n) = f(n-1) + f(n-2)
  - Tree
    - A tree has a root
    - Each child of the root is also a tree
- A problem of size n can be reduced to a problem of size less than n
  - Factorial: n → n-1
  - Sort: n → n-1
  - Binary search: n → n/2

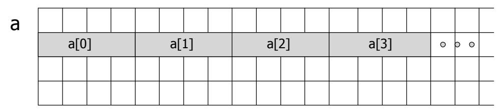
When Should We Use Recursion?

- When the homework problem says so
- When speed of code development takes precedence over code efficiency
- When the problem is naturally recursive
  - Fibonacci Series
- When the non-recursive solution is much harder
  - Hanoi tower
  - Solving maze

Arrays

- Name
- Type
- Length (or Size) – number of elements in the array
- Values

Computer Memory



Access Array Elements

- arrayname.length
- Index is from 0 to (arrayname.length-1)

```
int a[];
a = new int[10];

a[5] = 3;    // assign 3 to the 6th element

// prints out all elements
for( int i=0 ; i < a.length ; ++i )
  System.out.println( a[i] );
```

index

Array as Parameter

- Write a method sumArray() which returns the sum of the elements in a given array

```
int sumArray( ?? )
{
    int sum = 0;

    ??

    return sum;
}
```

Array as Return Type

- Write a method createArray() which returns an integer array of given size n.

```
?? createArray( int n )
{
    return ??;
}
```

Multidimensional Data

	HW0	HW1	HW2
Student1	90	80	100
Student2	80	75	85
Student3	70	90	70
Student4	50	50	80

Multidimensional Array

- Array of arrays
- Initialization??
- Allocation??

Multidimensional Array as Array-of-Arrays

```
int grades2 = { {10, 80, 100}, {10, 75, 85},
                {10, 90, 70}, {10, 50, 80} };
```

- grades2 – an array of 4 arrays
  - grades2[0] – an array of 3 integers: `10`, `80`, and `100`
  - …
  - grades2[3] – an array of 3 integers: `10`, `50`, and `80`

Arrayname.length and Friends

- arrayname.length is the length of the 1st dimension
- arrayname[i].length is the length of the 2nd dimension
- arrayname[i][j].length is the length of the 3rd dimension
- …

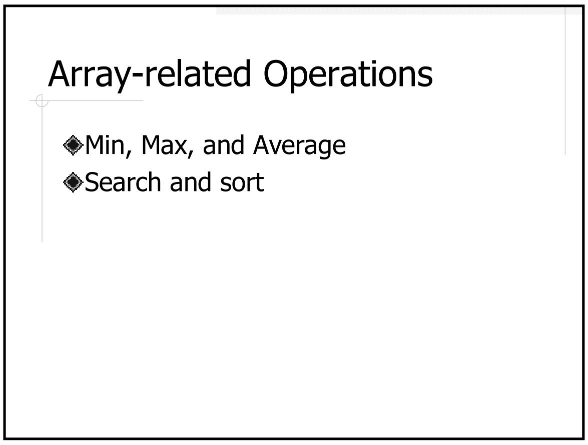
More Fun with Multidimensional Array

- Each row doesn't have to have the same number of elements

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

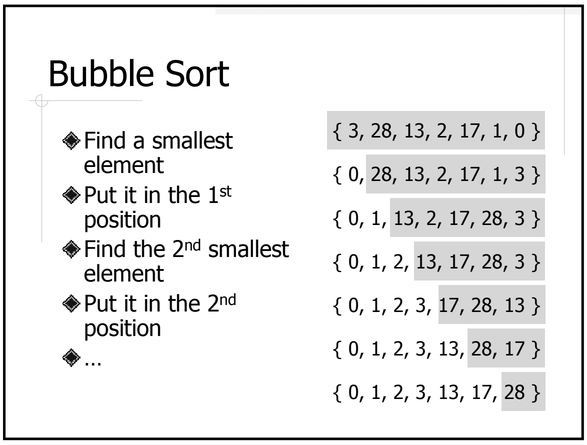
```
int a[][];

// allocation
a = new int[4][];
for( int i=0 ; i < a.length ; ++i )
  a[i] = new int[??];
```

- Exercise: add initialization to the code above



Array-related Operations

- Min, Max, and Average
- Search and sort



Bubble Sort

- Find a smallest element
- Put it in the 1st position
- Find the 2nd smallest element
- Put it in the 2nd position
- …

{ 3, 28, 13, 2, 17, 1, 0 }

{ 0, 28, 13, 2, 17, 1, 3 }

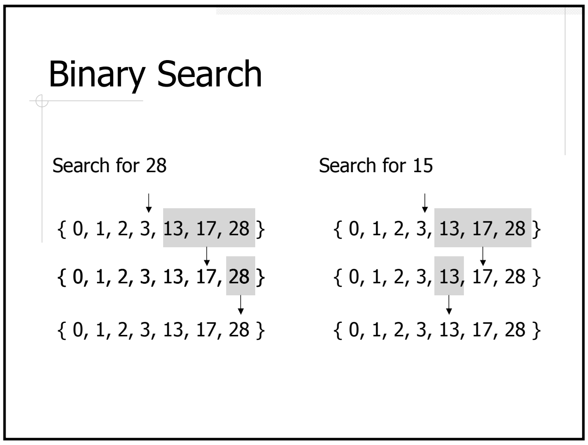
{ 0, 1, 13, 2, 17, 28, 3 }

{ 0, 1, 2, 13, 17, 28, 3 }

{ 0, 1, 2, 3, 17, 28, 13 }

{ 0, 1, 2, 3, 13, 28, 17 }

{ 0, 1, 2, 3, 13, 17, 28 }



Binary Search

Search for 28

{ 0, 1, 2, 3, 13, 17, 28 }

{ 0, 1, 2, 3, 13, 17, 28 }

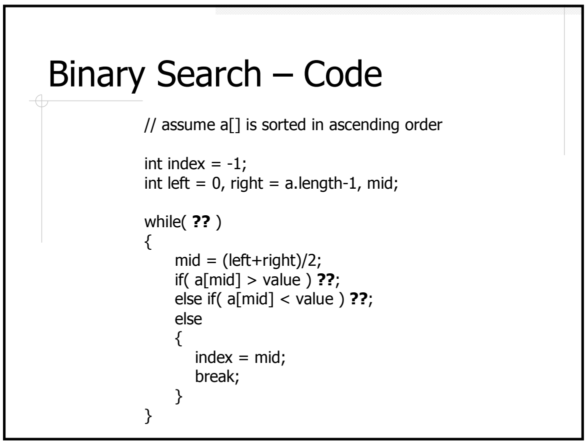
{ 0, 1, 2, 3, 13, 17, 28 }

Search for 15

{ 0, 1, 2, 3, 13, 17, 28 }

{ 0, 1, 2, 3, 13, 17, 28 }

{ 0, 1, 2, 3, 13, 17, 28 }



Binary Search – Code

```
// assume a[] is sorted in ascending order

int index = -1;
int left = 0, right = a.length-1, mid;

while( ?? )
{
    mid = (left+right)/2;
    if( a[mid] > value ) ??;
    else if( a[mid] < value ) ??;
    else
    {
        index = mid;
        break;
    }
}
```