# AP Computer Science A

Java Programming Essentials [Ver. 3.0]

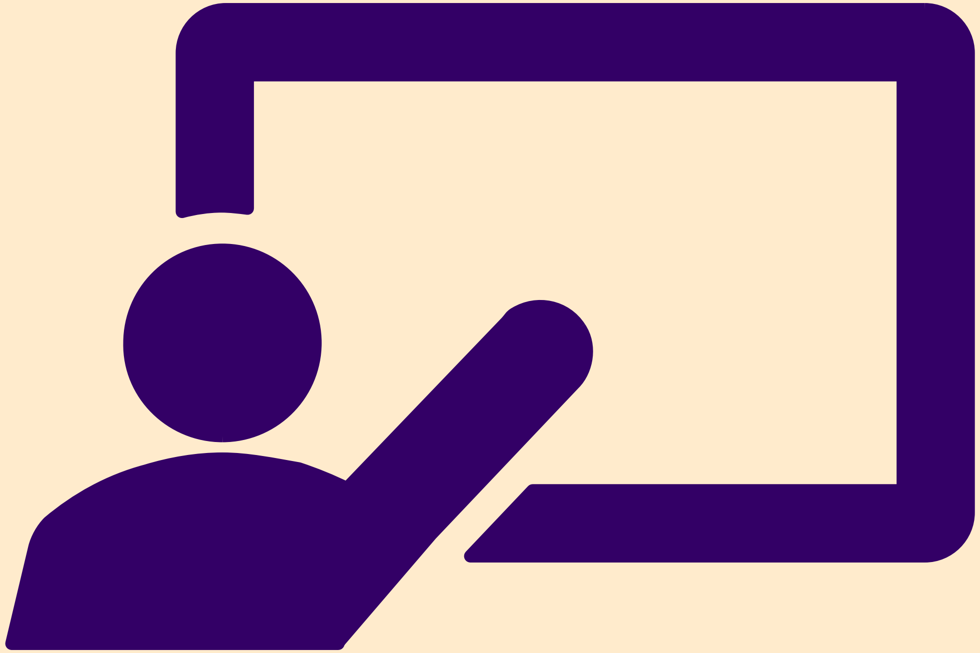
## Unit 2: Structured Programming

CHAPTER 4B: BOOLEAN ALGEBRA

DR. ERIC CHOU IEEE SENIOR MEMBER

# Objectives

- Basic Boolean Algebra
- Logic Design
- Number Line Analysis
- Aggregated Logic Function (hasEven, allEven)
- Calendar Year
- Software Design Life Cycle
- Top-down Design Bottom-Up Implementation
- Pseudo Code
- PrintCalendar Project



# Basic Boolean Algebra

LECTURE 1

# Introduction

- The most **obvious** way to **simplify** Boolean expressions is to manipulate them in the same way as normal **algebraic expressions** are manipulated.
- With regards to logic relations in digital forms, a set of rules for symbolic manipulation is needed in order to solve for the unknowns.

# Introduction

- A set of rules formulated by the English mathematician George Boole describe certain propositions whose outcome would be either true or false.
- With regard to digital logic, these rules are used to describe circuits whose state can be either, 1 (**true**) or 0 (**false**). In order to fully understand this, the relation between the AND gate, OR gate and NOT gate operations should be appreciated. A number of rules can be derived from these relations as Table 1 demonstrates.

# Introduction

- A number of rules can be derived from these relations as Table 1 demonstrates.

  - **Rule 1: X = 0 or X = 1**
  - **Rule 2: 0 * 0 = 0**
  - **Rule 3: 1 + 1 = 1**
  - **Rule 4: 0 + 0 = 0**
  - **Rule 5: 1 * 1 = 1**
  - **Rule 6: 1 * 0 = 0 * 1 = 0**
  - **Rule 7: 1 + 0 = 0 + 1 = 1**

**Table 1: Boolean Postulates**

# Laws of Boolean Algebra

- Table 2 shows the basic Boolean laws.
- Note that every law has two expressions, (a) and (b). This is known as *duality*.
- These are obtained by changing every AND(*) to OR(+), every OR(+) to AND(*) and all 1's to 0's and vice-versa.
- It has become conventional to drop the * (AND symbol) i.e. A.B is written as AB.

# Boolean Algebra Laws

**T1 : Commutative Law**

(a) $A + B = B + A$

(b) $A\,B = B\,A$

**T2 : Associate Law**

(a) $(A + B) + C = A + (B + C)$

(b) $(A\,B)\,C = A\,(B\,C)$

**T3 : Distributive Law**

(a) $A\,(B + C) = A\,B + A\,C$

(b) $A + (B\,C) = (A + B)\,(A + C)$

**T4 : Identity Law**

(a) $A + A = A$

(b) $A\,A = A$

**T5 :**

(a) $AB + A\overline{B} = A$

(b) $(A+B)(A+\overline{B}) = A$

**T6 : Redundance Law**

(a) $A + A\,B = A$

(b) $A\,(A + B) = A$

**T7 :**

(a) $0 + A = A$

(b) $0\,A = 0$

**T8 :**

(a) $1 + A = 1$

(b) $1\,A = A$

**T9 :**

(a) $\overline{A} + A = 1$

(b) $\overline{A}\,A = 0$

**T10 :**

(a) $A + \overline{A}\,B = A + B$

(b) $A\,(\overline{A} + B) = A\,B$

**T11 : De Morgan's Theorem**

(a) $(\overline{A + B}) = \overline{A}\;\overline{B}$

(b) $(\overline{A\,B}) = \overline{A} + \overline{B}$

# True Table for Logic Functions

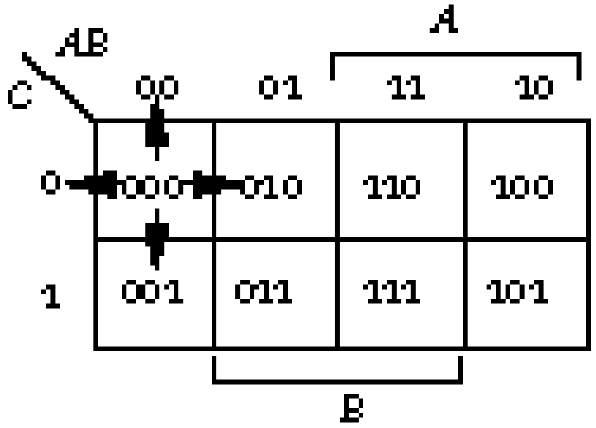
<expr A operator B> can have 16 possible outcomes which equals $2^4$ (4 A/B combinations each has 2 possible outcome)

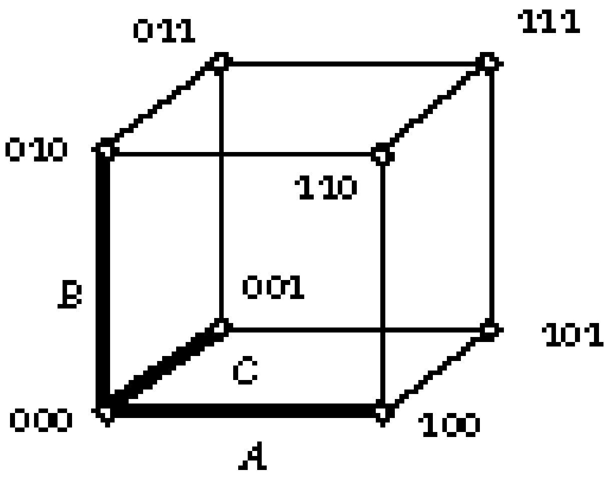
(Operators)

| INPUT AB | ZERO | AND | A > B | A | B > A | B | XOR | OR | NOR | XNOR | NOT 'B' | A ≤ B | NOT 'A' | B ≤ A | NAND | ONE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 01 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 10 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 11 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

# Boolean Cube and Theorem of Symmetry

## The dual of any Boolean property




If Boolean Expression

T(A, B, +, *, =, 1, 0) is valid.

T(A, B , *, +, = 0, 1) is also valid.

A B + A = A

(A + B) * A = A

A + (-A) = 1

A * (-A) = 0

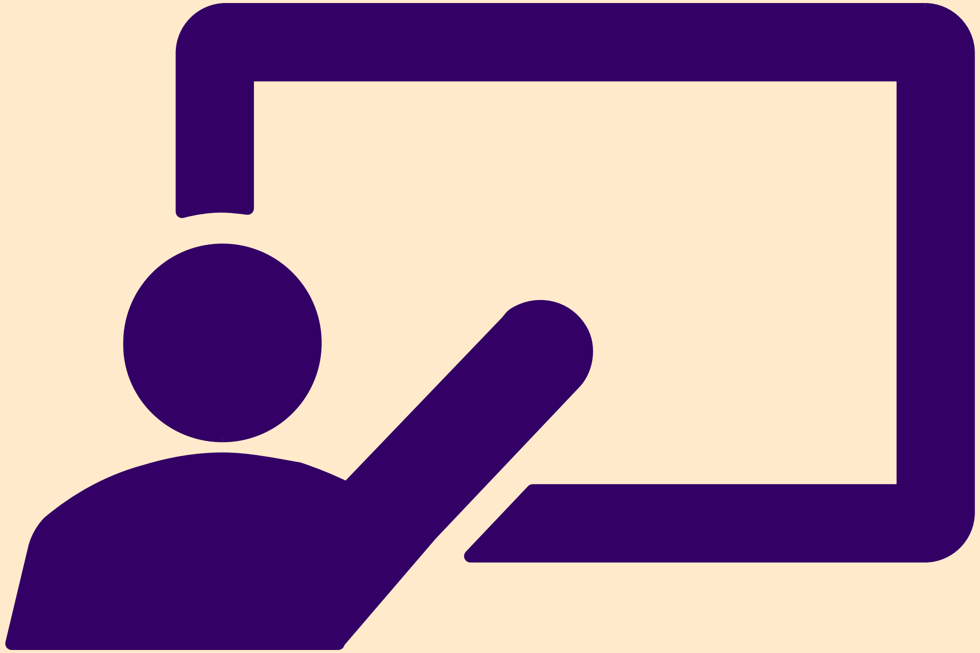
# Example of De Morgan's Theorem

if ((x % 5 != 0) && (x % 2 != 0))        // x is not multiple of 5 and x is not multiple of 2.

equivalent to

if ( !((x % 5 == 0) || (x % 2 == 0)) )   //  x is not (multiple of 5 or 2)

Download and work on BooleanQuiz.pdf



# Logic Design

LECTURE 2

# Student GPA Issues 1:

## (smaller set should go first)

**Smaller set go first:**

```
if (math>=90)        {mathGrade = 'A';}
else if (math>=80)  {mathGrade = 'B';}
else if (math>=70)  {mathGrade = 'C';}
else if (math>=60)  {mathGrade = 'D';}
else                       {mathGrade = 'F';}
```

**Smaller set go first:**

```
if (math<60)          {mathGrade = ‘F';}
else if (math>=60){mathGrade = ‘D';}
else if (math>=70){mathGrade = 'C';}
else if (math>=80){mathGrade = ‘B';}
else if (math>=90){mathGrade = ‘A';}
```

# Student GPA Issue 2:
## (Corner Cases)

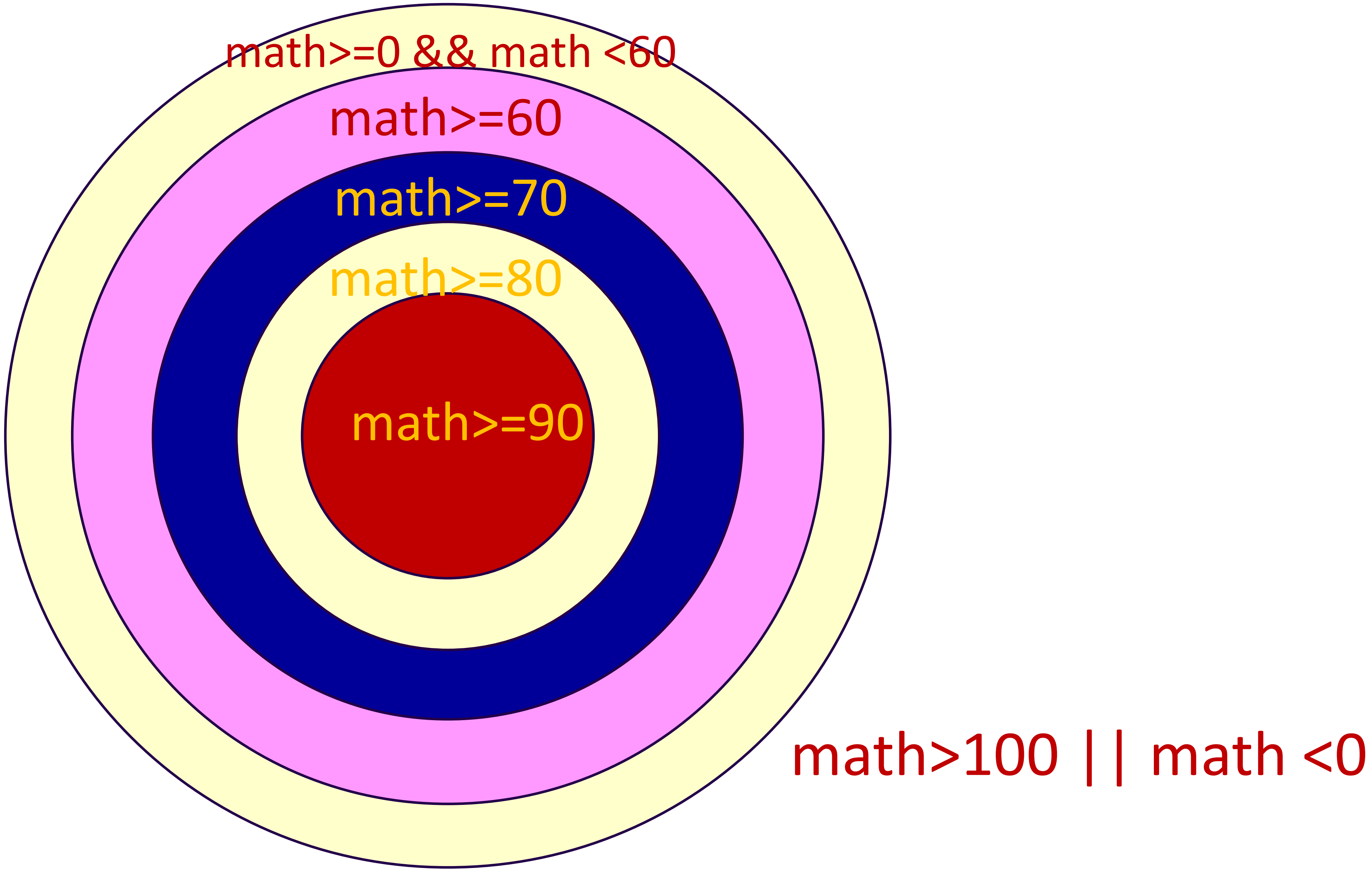
**Smaller set go first:**

```
if (math>=90)           {mathGrade = 'A';}
else if (math>=80)  {mathGrade = 'B';}
else if (math>=70)  {mathGrade = 'C';}
else if (math>=60)  {mathGrade = 'D';}
else                         {mathGrade = 'F';}
```

**Corner Cases Handled:**

```
if (math>=90 && math<=100)
{mathGrade = 'A';}
 else if (math>=80 && math<=100)
{mathGrade = 'B';}
 else if (math>=70 && math<=100)
{mathGrade = 'C';}
 else if (math>=60 && math<=100)
{mathGrade = 'D';}
 else if (math>= 0 && math<=100)
{mathGrade = 'F';}
 else {mathGrade = ‘N’; }  // not graded yet
```

# Venn Diagram for **StudentGPA** series programs



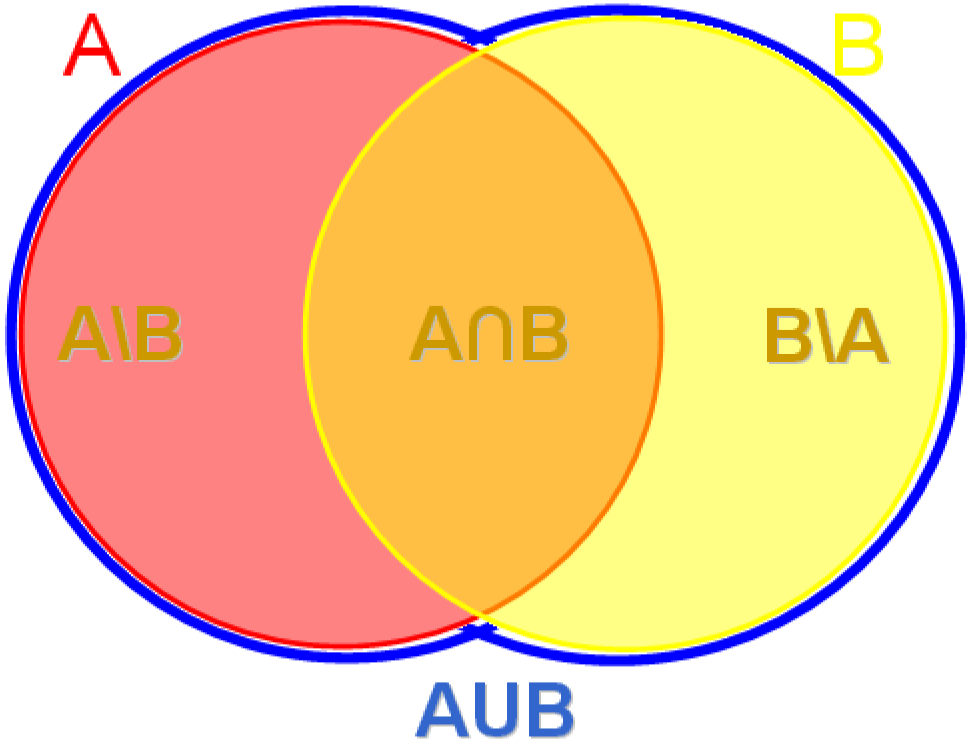
# Set Theory and Logic Design (only positive numbers are shown)

boolean m5 = ( x % 5 == 0);  // Set S5 = [0, 5, 10, 15, 20, …]

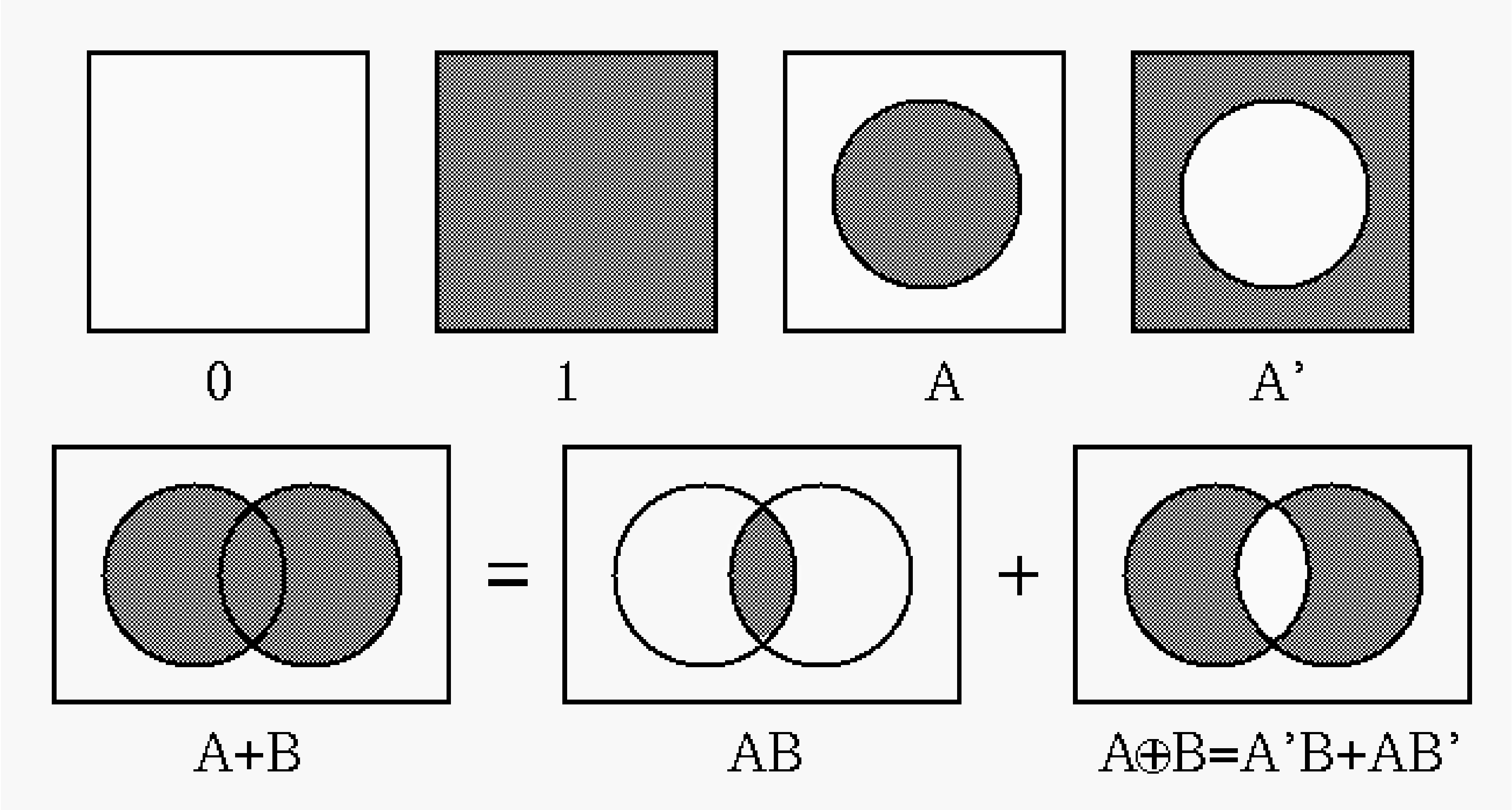
boolean m2 = ( x % 2 == 0);  // Set S2 = [0, 2, 4, 6, 8, 10, …]

| Boolean Expression | Set | Set Components |
|---|---|---|
| m5 && m2 | $S5 \cap S2$ | [0, 10, 20, 30, 40, …] |
| m5 \|\| m2 | $S5 \cup S2$ | [0, 2, 4, 5, 6, 8, 10, 12, 14, 15, …] |
| m2 && !m5 | $S2 - S5$ | [2, 4, 6, 8, 12, 14, 16, 18, …] |
| m5 && !m2 | $S5 - S2$ | [5, 15, 25, 35, 45, …] |
| m2 ^ m5 | $S5 \oplus S2$ | [2,4, 5, 6, 8, 12, 14, 15, 16, …] |

# Venn Diagram Analysis



# Venn Diagram Analysis



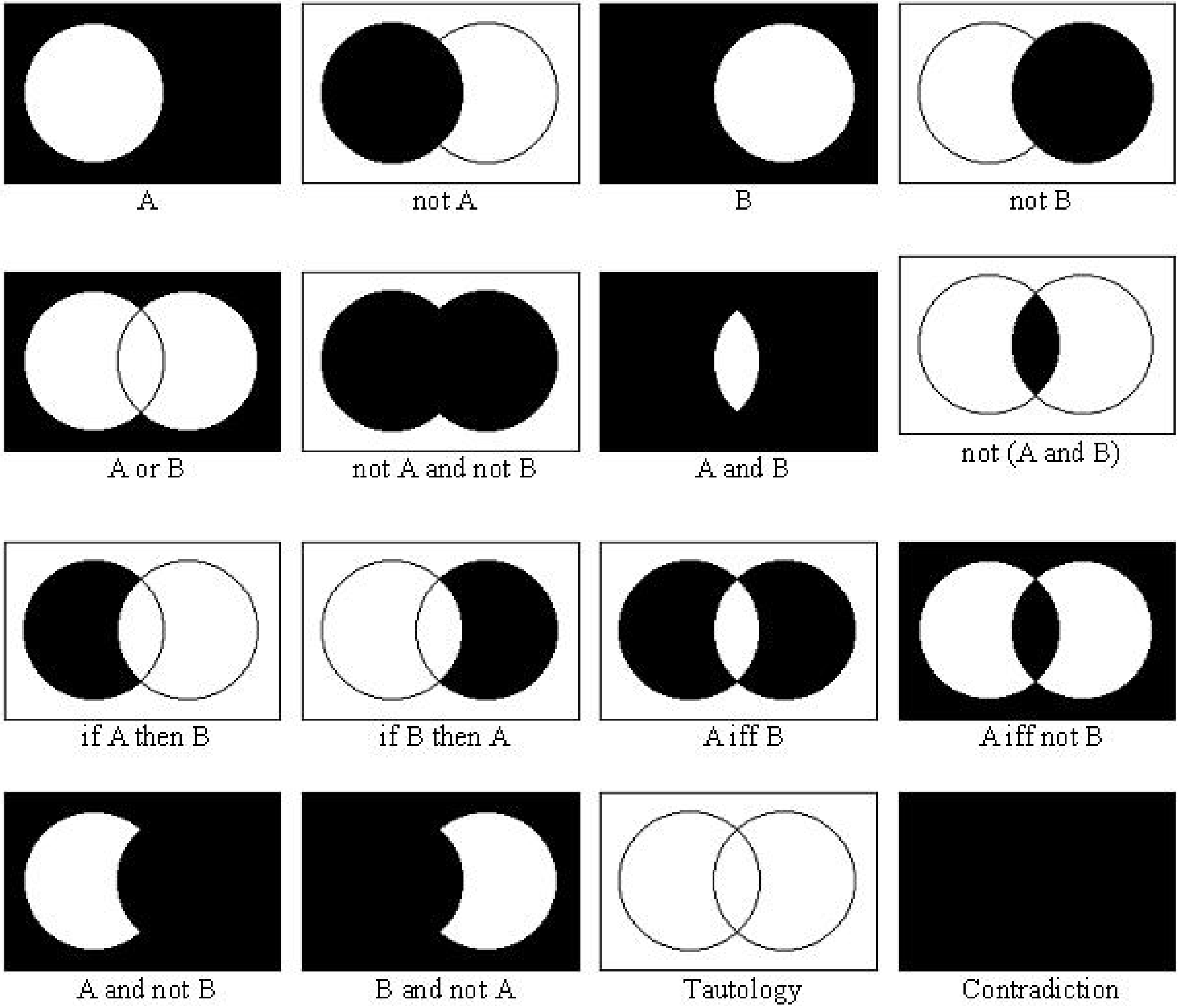
# All 16 Boolean Functions can be expressed by (x, y, !, &&, ||)

| | | | | |
|---|---|---|---|---|
| x | 0 | 0 | 1 | 1 |
| y | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 |
| $x \cdot y$ | 0 | 0 | 0 | 1 |
| $x \cdot y'$ | 0 | 0 | 1 | 0 |
| $x$ | 0 | 0 | 1 | 1 |
| $x' \cdot y$ | 0 | 1 | 0 | 0 |
| $y$ | 0 | 1 | 0 | 1 |
| $x \cdot y' + x' \cdot y$ | 0 | 1 | 1 | 0 |
| $x+y$ | 0 | 1 | 1 | 1 |
| $(x+y)'$ | 1 | 0 | 0 | 0 |
| $x \cdot y + x' \cdot y'$ | 1 | 0 | 0 | 1 |
| $y'$ | 1 | 0 | 1 | 0 |
| $x+y'$ | 1 | 0 | 1 | 1 |
| $x' \cdot y$ | 1 | 1 | 0 | 0 |
| $x'+y$ | 1 | 1 | 0 | 1 |
| $(x \cdot y)'$ | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

# All 16 Boolean Functions can be expressed by (x, y, !, &&, ||)

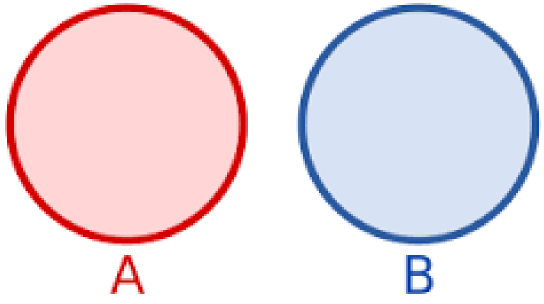
| Boolean Functions in Java | |
|---|---|
| boolean f0 = false; | boolean f8 = !(x+y); |
| boolean f1 = x && y; | boolean f9 = x && y + !x && !y; |
| boolean f2 = x && !y; | boolean f10 = !y; |
| boolean f3 = x; | boolean f11 = x + !y; |
| boolean f4 = !x && y; | boolean f12 = !x && y; |
| boolean f5 = y; | boolean f13 = !x + y; |
| boolean f6 = x && !y + !x && y; | boolean f14 = !(x && y) |
| boolean f7 = x + y; | boolean f15 = true; |

# Advanced Venn Diagram for all 16 Logic Functions for Two Inputs *f(A, B)*



# Mutual Exclusive Sets

```
boolean male   = (gender == ‘M’);
boolean female = (gender == ‘F’);
```




if (male) { // all male get here , and all female will not get here}

if (female) { // all female get here, and all male will not get here}

# Set Contained by Another Set

Contained by (A == (x>2),  B == (x>1))

if a is true then b must be true   ==  !a || b

If x > 2 then x > 1 == !(x > 2) || (x>1)    (x>2) is contained by (x>1)

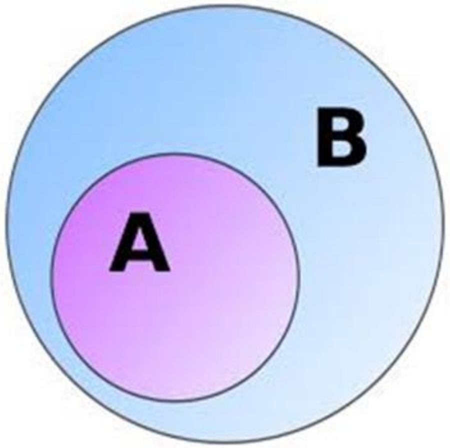
If you want to have A test first and B-A test:

```
if (x > 2) { // all (x>2) get here }
else if (x > 1) { // only x>1 && !(x>2) get here }
```

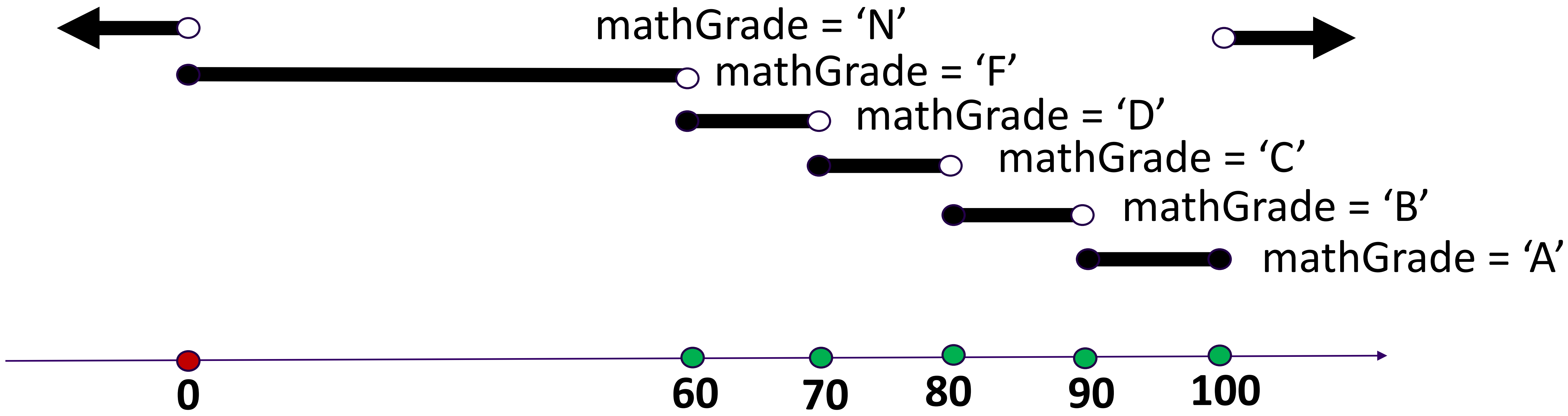
If you want to have A test and then B test:

```
if (x>2) { // all x > 2 get here }
if (x>1) { // all x > 1 get here }
```

Don’t try this:  if (x>1) { // all x>1 get here } else if (x>2) { // no x can get here}



# Number Line Analysis (Letter Grade)



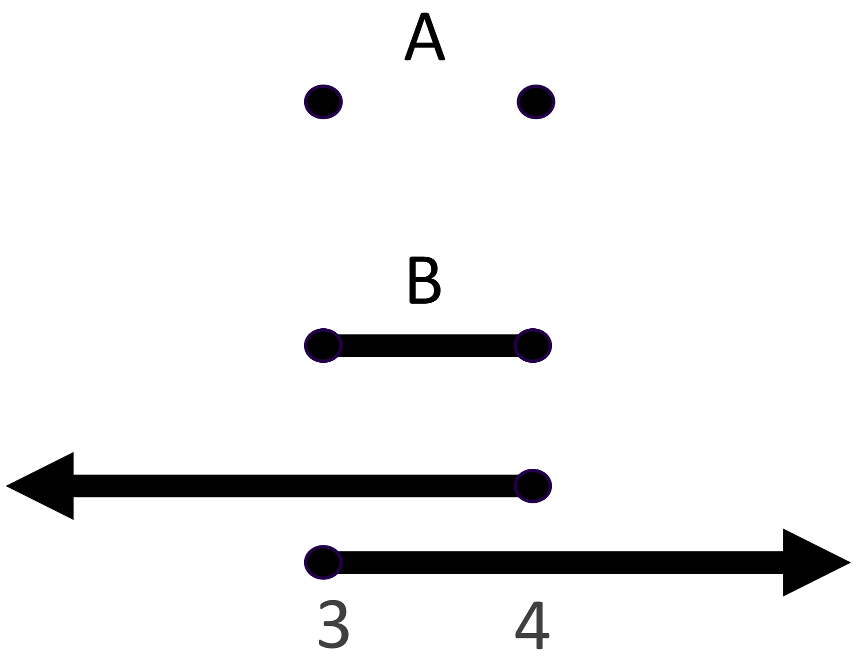
# Sometimes the Logic Design Result Also Depends on the Data Type as Well

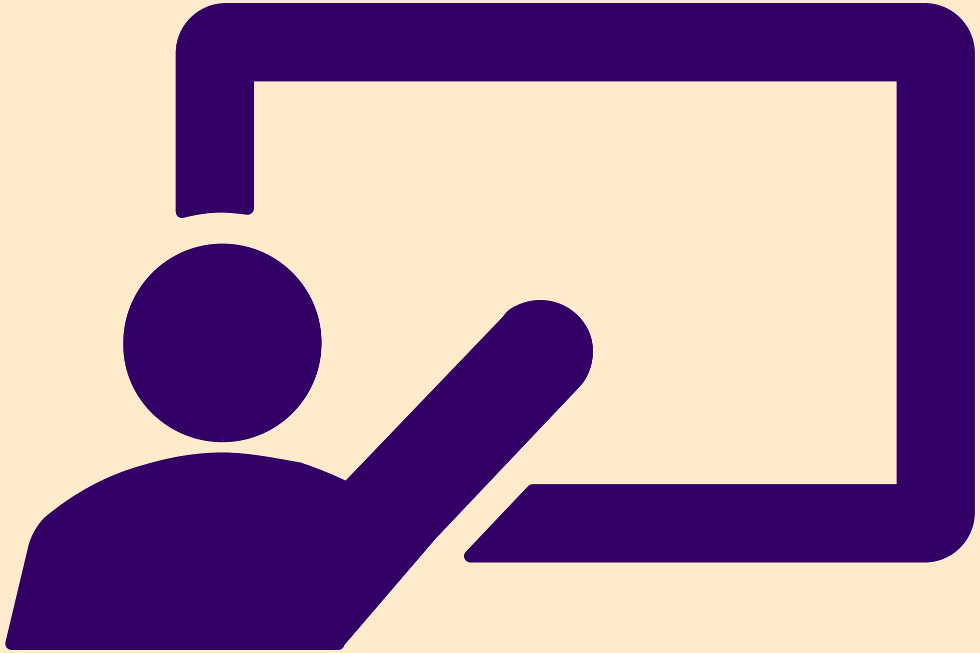
```
if (x == 3 ||  x == 4) { // set A }
if (x >= 3 && x <= 4) { // set B }
```

For int data type, A is equivalent to B

For double data type, A is not equivalent to B

for example, x = 3.5 is in B but not in A





# hasEven, AllEven

LECTURE 3

# isEven – Full Logic

```
public static boolean isEven1(int n){
        if (n%2==0){
            return true;
         }
        else {
            return false;
         }
    }
```

# isEven – Exit by Return

```
public static boolean isEven2(int n){
        if (n%2==0){
            return true;
        }
        // n%2!=0 after this line
        return false;
    }
```

# isEven – Exit by Return

```
public static boolean isEven3(int n){
        return n%2==0;
    }
```

# Logic (Quantifier)

- $\forall$ x, x < 3 → for all x, x < 3
- $\exists$ x, x < 3 → Exists x, x < 3

# Logic (Quantifier)

- $\forall$ x, x % 2 == 0 → for all x, x is even

  $\neg\forall$ x, x % 2 == 0 →Not( for all x, x is even)

  → Exist x, x is odd

  →$\exists$ x, x % 2 != 0

# Logic (Quantifier)

- $\exists$ x, x % 2 == 0 → Exists x, x is even
- $\neg(\exists$ x, x % 2 == 0) → not (Exists x, x is even)

  → for all x, x is odd

  → $\forall$ x, x % 2 != 0

```java
static int[] a = {1, 2, 3, 4, 5};
static int[] b = {1, 3, 5, 7, 9};
static int[] c = {2, 4, 6, 8, 10};

public static boolean allEven(int[] x){
   for (int i=0; i<x.length; i++){
       if (x[i]%2 !=0) return false;
    }
   return true;
}
public static boolean allOdd(int[] x){
   for (int i=0; i<x.length; i++){
       if (x[i]%2 ==0) return false;
    }
   return true;
}
public static boolean hasEven(int[] x){
   for (int i=0; i<x.length; i++){
       if (x[i]%2 ==0) return true;
    }
   return false;
}
public static boolean hasOdd(int[] x){
   for (int i=0; i<x.length; i++){
       if (x[i]%2 !=0) return true;
    }
   return false;
}
```

```java
public static void main(String[] args){
    System.out.print("\f");
    System.out.printf("a is allEven is %b\n", allEven(a));
    System.out.printf("a is allOdd is %b\n",  allOdd(a));
    System.out.printf("a is hasEven is %b\n", hasEven(a));
    System.out.printf("a is hasOdd is %b\n",  hasOdd(a));
    System.out.println();
    System.out.printf("b is allEven is %b\n", allEven(b));
    System.out.printf("b is allOdd is %b\n",  allOdd(b));
    System.out.printf("b is hasEven is %b\n", hasEven(b));
    System.out.printf("b is hasOdd is %b\n",  hasOdd(b));
    System.out.println();
    System.out.printf("c is allEven is %b\n", allEven(c));
    System.out.printf("c is allOdd is %b\n",  allOdd(c));
    System.out.printf("c is hasEven is %b\n", hasEven(c));
    System.out.printf("c is hasOdd is %b\n",  hasOdd(c));
    System.out.println();
}
```

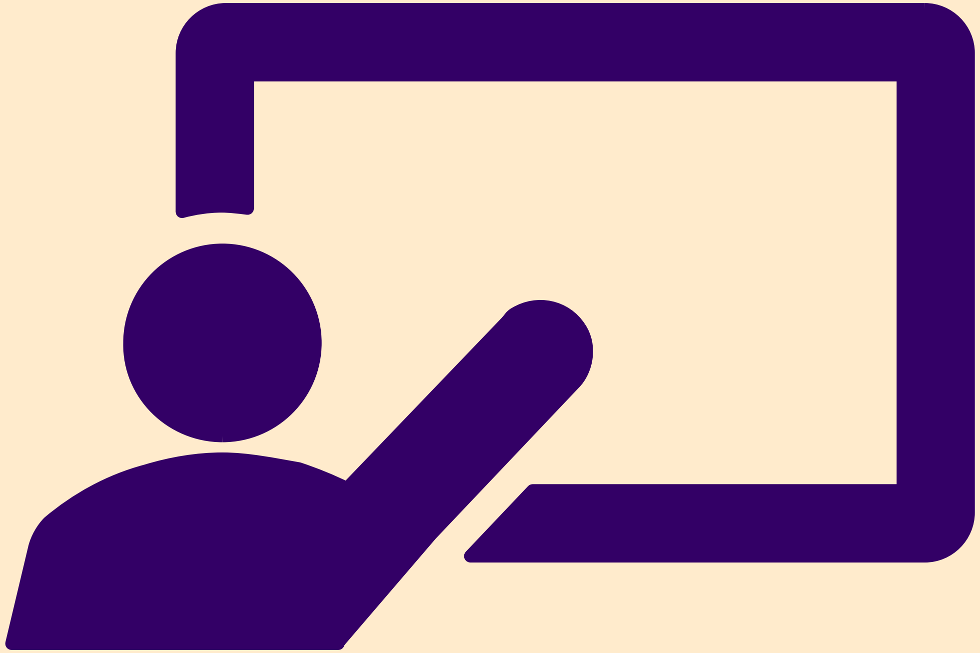
```
a is allEven is false
a is allOdd is false
a is hasEven is true
a is hasOdd is true

b is allEven is false
b is allOdd is true
b is hasEven is false
b is hasOdd is true

c is allEven is true
c is allOdd is false
c is hasEven is true
c is hasOdd is false
```

# Demonstration Program

ALLEVEN.JAVA



# Calendar Year

LECTURE 4

# Lab

CALENDAR YEAR

LEAPYEAR.JAVA

# Background Information: (Leap Year)

- Which Years are Leap Years?
- In the Gregorian calendar 3 criteria must be taken into account to identify leap years:

- The year is evenly divisible by 4;
- If the year can be evenly divided by 100, it is NOT a leap year, unless;
- The year is also evenly divisible by 400. Then it is a leap year.

- The year 2000 was somewhat special as it was the first instance when the third criterion was used in most parts of the world since the transition from the Julian to the Gregorian Calendar.

Year of the Rat

Year of the Ox

Year of the Tiger

Year of the Rabbit

Year of the Dragon

Year of the Snake

Year of the Horse

Year of the Sheep

Year of the Monkey

Year of the Rooster

Year of the Dog

Year of the Pig

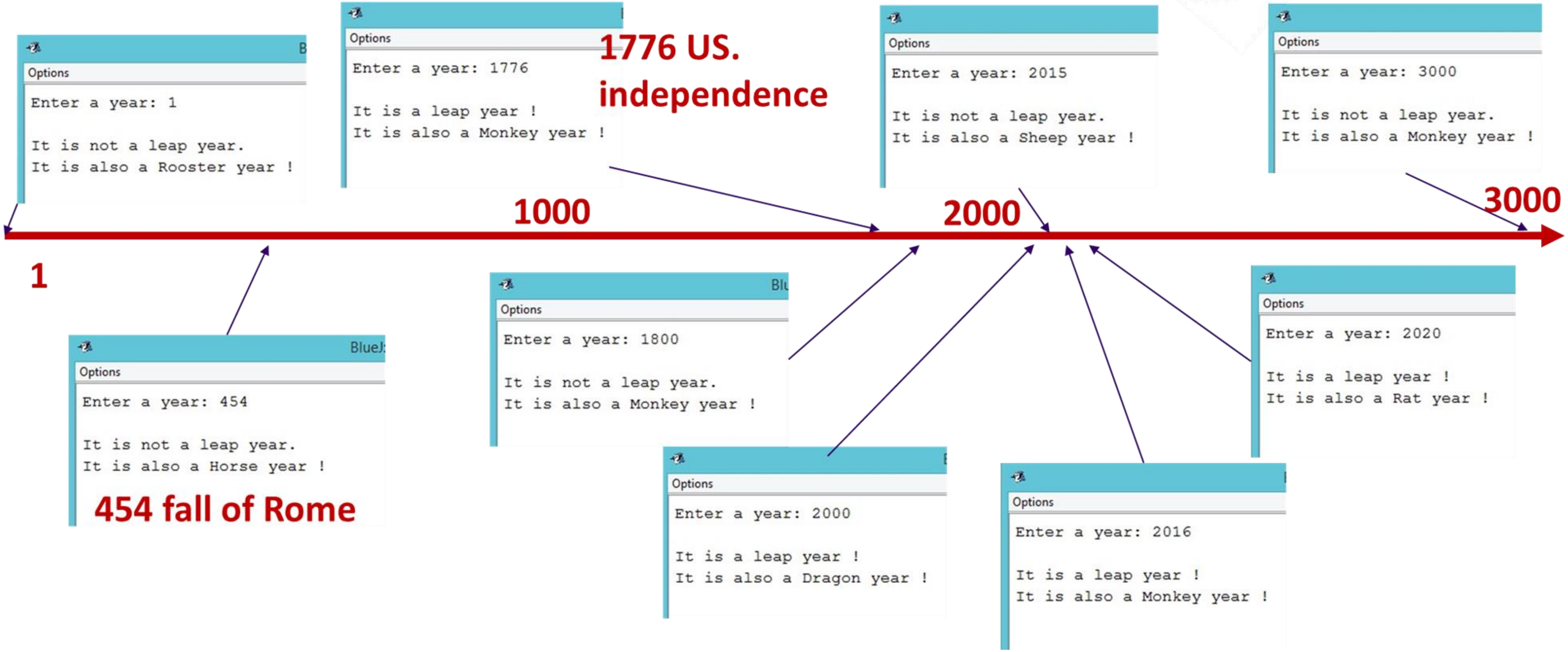
# Background Information:
## (Chinese Zodiac)

- Jupiter goes around the sun every 11.87 years. (approximately 12 years.) Chinese call it ***Planet of Years***. They use it to calculate for the Zodiac. Every 12 years is one rotation.
- The year 1948 is year of Rat. The year 1950 is year of Tiger. The year 2015 is year of Sheep (sometimes also called goat/ram. Chinese believe they are in one category. )
- If the year 2015 is 67 (2015-1948) years away from 1948 and 67 % 12 is 7, 7 years away from rat year is year of Sheep.
- So, you may use y = (x-1948) % 12 to know that Year x is y years away from Rat year. Then, look up from the table.

# Lab: Calendar Year (LeapYear.java)

- Write a program to ask the calendar year between 1948 and now, to determine 2 things:
  - (1) Is it a leap year?
  - (2) What Chinese Zodiac Year it is?
- Then, Print out a Calendar Year Report with these information.

# Expected Result:



# Demonstration Program

CHINESEZODIAC.JAVA

# Software Design Life Cycle

LECTURE 5

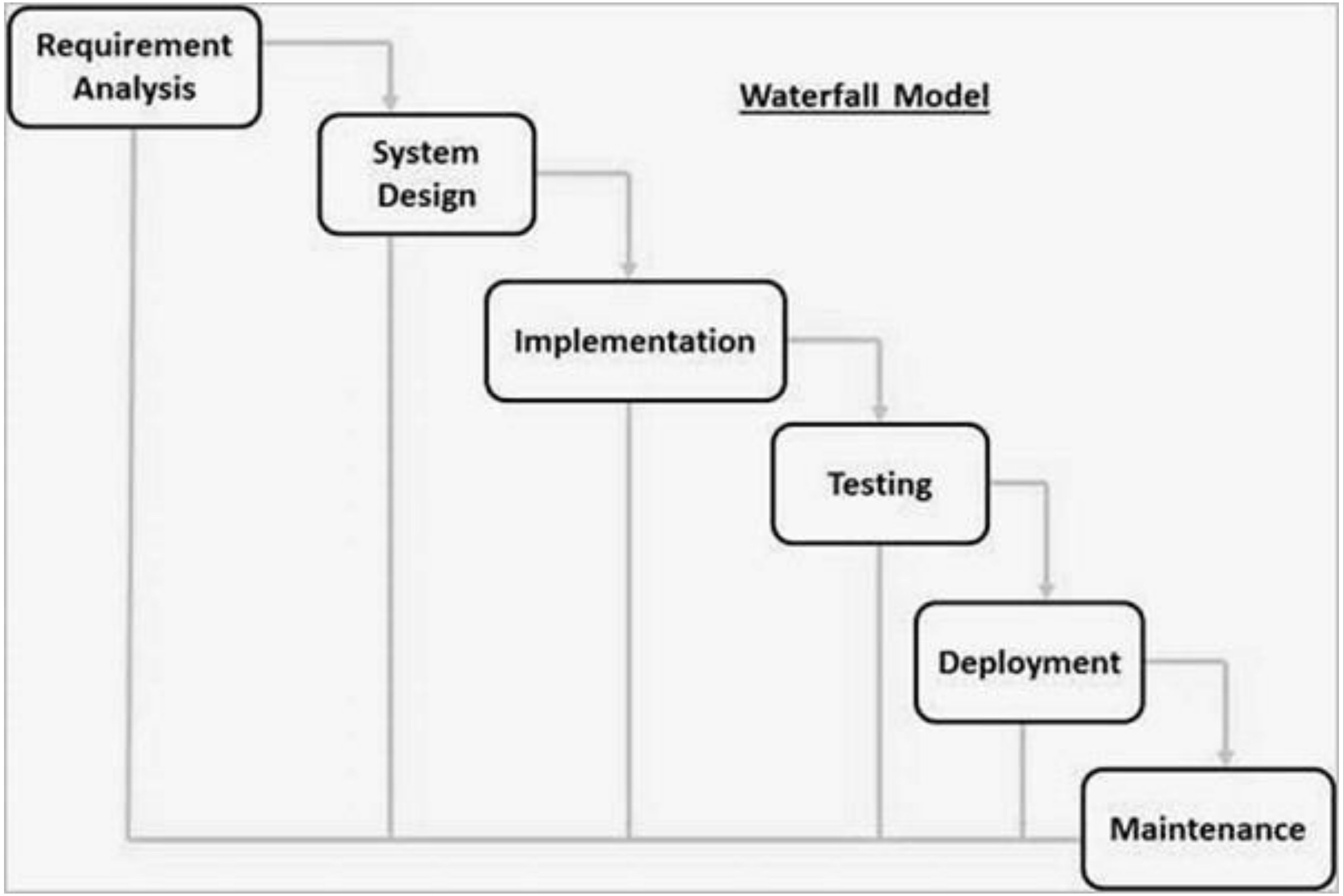
# Software Engineering

- Software engineering is the branch of computer science that deals with the design, development, testing, and maintenance of software applications.
- Software engineers apply engineering principles and knowledge of programming languages to build software solutions for end users.


SOFTWARE ENGINEERING
BUGS
DOWNLOAD
APPLICATION
INSTALL
LANGUAGE
NETWORK
SCRIPT
UPDATE
ComputerHope.com

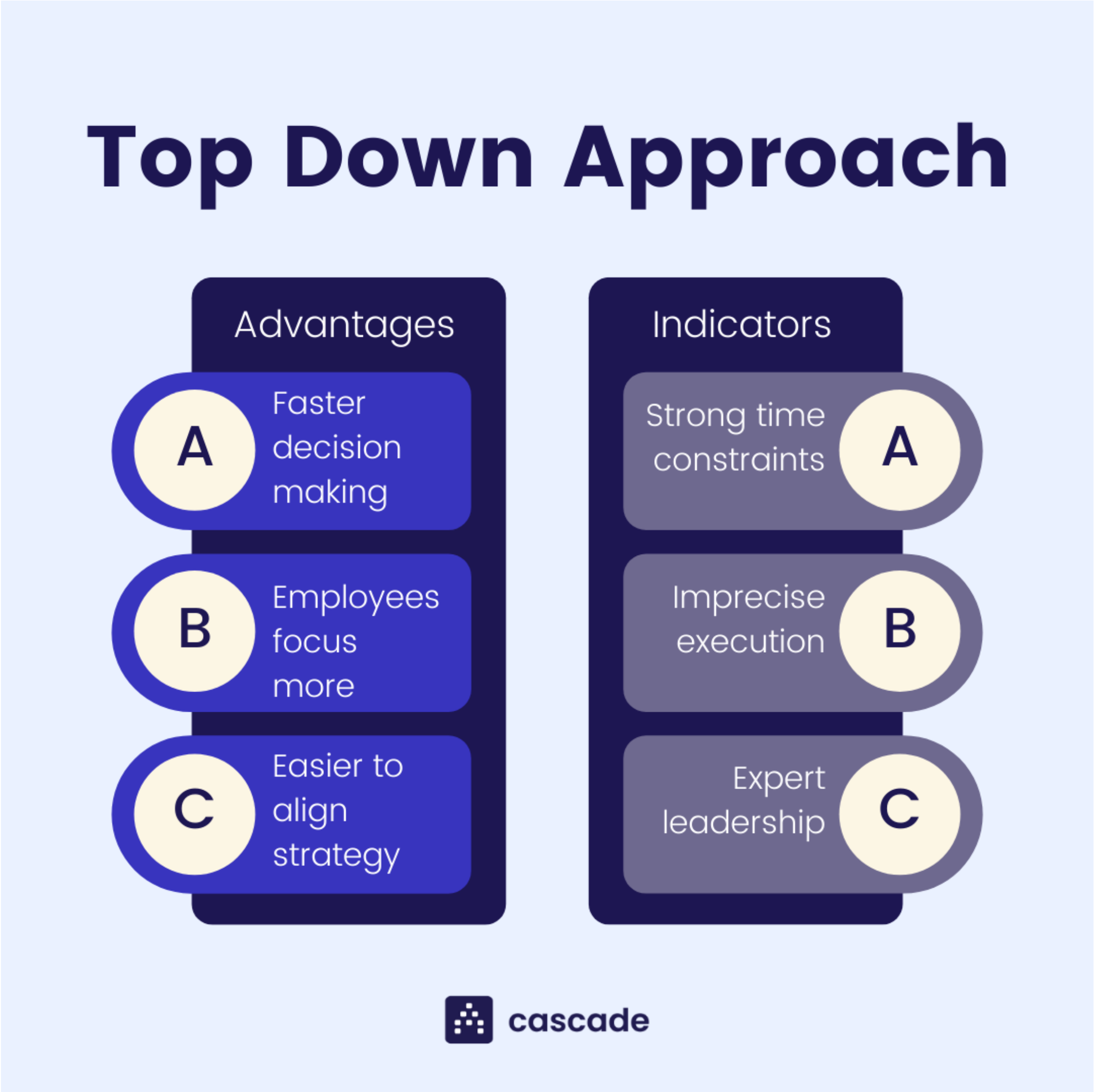
# Software Design Life Cycle

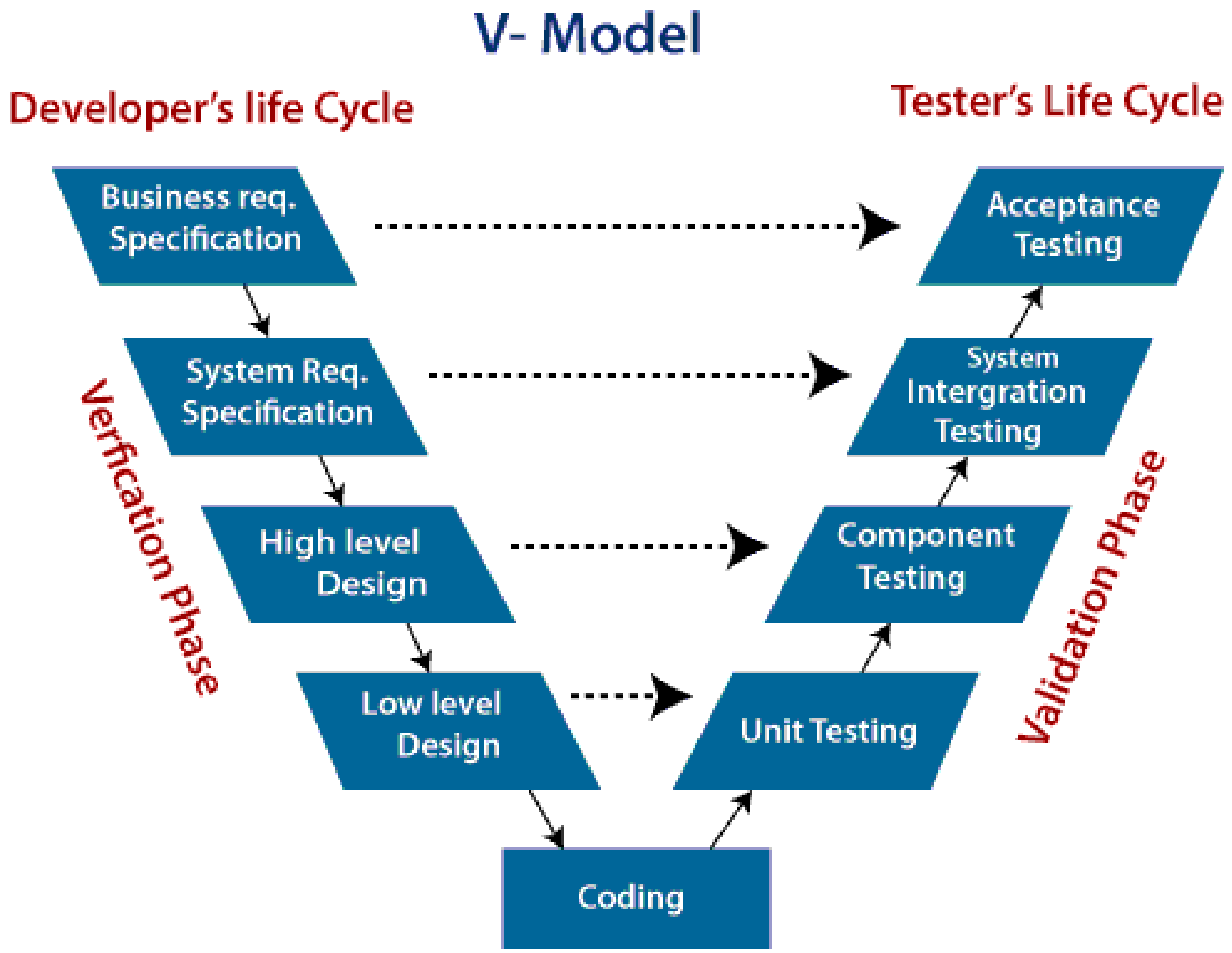
- The Software Development Life Cycle (SDLC) refers to a methodology with clearly defined processes for creating high-quality software. in detail, the SDLC methodology focuses on the following phases of software development:
  - Requirement analysis
  - Planning
  - Software design such as architectural design
  - Software development
  - Testing
  - Deployment


Waterfall Model
Requirement Analysis
System Design
Implementation
Testing
Deployment
Maintenance

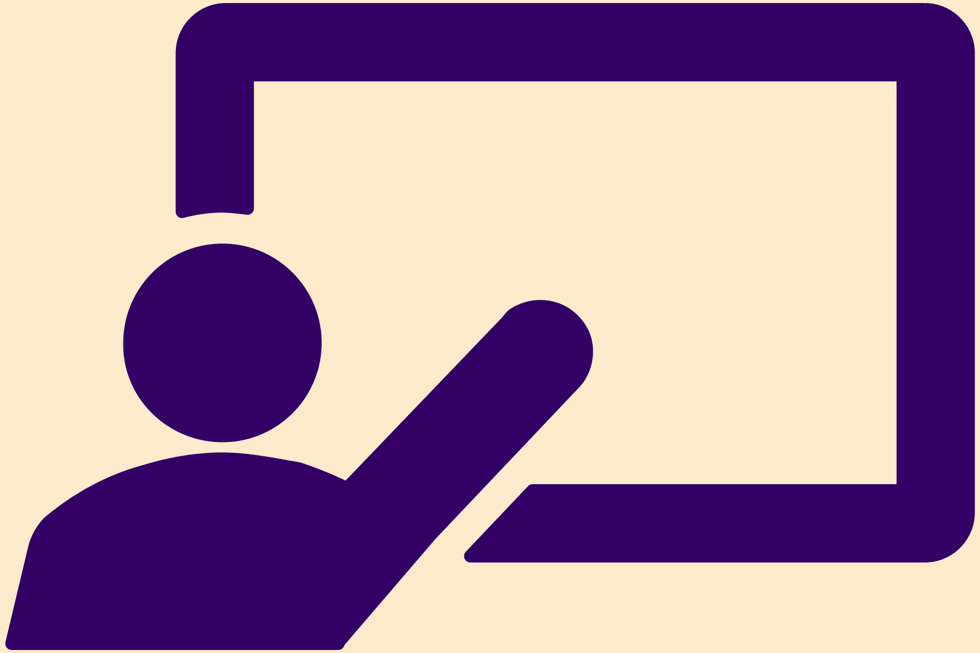
# Top-down versus Bottom Up

- Top down approach starts with the big picture, then breaks down from there into smaller segments.
- A bottom-up approach is the piecing together of systems to give rise to more complex systems, thus making the original systems sub-systems of the emergent system.


Top Down Approach
Advantages
A
Faster decision making
B
Employees focus more
C
Easier to align strategy
Indicators
Strong time constraints
A
Imprecise execution
B
Expert leadership
C
cascade




# Top-Down Design and Bottom-Up Implementation



# Pseudo Code

LECTURE 6

# What is pseudo code?

**Simple definition:**

Use **English** or **English-like** language to describe a program for a computation.

**Formal Definition:**

Pseudocode is an **artificial** and **informal** language that helps programmers develop algorithms. **Pseudocode** is a "text-based" detail (algorithmic) design tool.

The rules of Pseudocode are reasonably straightforward. All statements showing "dependency" are to be indented. These include while, do, for, if, switch.

# Pseudo Code Standard

http://www.engr.sjsu.edu/bjfurman/courses/ME30/ME30pdf/Notes_on_Algorithms.pdf

It does not need a standard because it is free style.

But if you need some reference, this one can be a good reference source.

# Method for Developing an Algorithm

1. **Define the problem**: State the problem you are trying to solve in clear and concise terms.

2. **List the inputs (information needed to solve the problem) and the outputs** (what the algorithm will produce as a result)

3. **Describe the steps** needed to convert or manipulate the inputs to produce the outputs. Start at a high level first, and keep refining the steps until they are effectively computable operations.

4. **Test the algorithm**: choose data sets and verify that your algorithm works!

# Pseudocode (or Program Design Language)

- Consists of natural language-like statements that precisely describe the steps of an algorithm or program
- **Statements** describe actions
- Focuses on the logic of the **algorithm** or program
- **Avoids** language-specific elements
- Written at a level so that the desired programming code can be generated almost automatically from each statement.
- Steps are numbered. Subordinate numbers and/or indentation are used for dependent statements in selection and repetition structures.

# Pseudocode Language Constructs

- <u>Computation/Assignment</u>
  - **Compute** var1 as the sum of x and y
  - **Assign** expression to var2
  - **Increment** counter1

- <u>Input/Output</u>
  - Input: **Get** var1, var2, …
  - Output: **Display** var1, var2, …

# Pseudocode Language Constructs

- <u>Selection</u>

<u>Single-Selection IF</u>

1. **IF** condition **THEN** (IF condition is true, then do subordinate statement 1, etc. If condition is false, then skip statements)
    1.1 statement 1
    1.2 etc.

# Pseudocode Language Constructs

<u>Double-Selection IF</u>

2. **IF** *condition* **THEN** (IF condition is true, then do subordinate statement 1, etc. If condition is false, then skip statements and execute statements under ELSE)
   2.1 statement 1
   2.2 etc.
3. **ELSE** (else if condition is not true, then do subordinate statement 2, etc.)
   3.1 statement 2
   3.2 statement 3
4. **SWITCH** expression **TO**
   4.1 case 1: action1
   4.2 case 2: action2
   4.3 etc.

4.4 default: actionx

# Pseudocode Language Constructs

- Repetition

  5. **WHILE** condition (while condition is true, then do subordinate statements)

     5.1 statement 1

     5.2 etc.

  DO – WHILE structure (like WHILE, but tests condition at the end of the loop. Thus, statements in the structure will always be executed at least once.)

  6. **DO**

     6.1 statement 1

     6.2 etc.

  7. **WHILE** condition

# Pseudocode Language Constructs

FOR structure (a specialized version of WHILE for repeating execution of statements a specific number of times)

8. **FOR** bounds on repetition
    - 8.1 statement 1
    - 8.2 etc.

# Pseudo Code Example 1: Plain English

**To find maximum from a group of data:**

set the maximum to one of the element.

each time compare the maximum to one of the rest of elements in the group.

If the element picked is larger than our maximum

Then, set the element to be the new maximum

look at the next element to repeat the comparison

After all of the elements have been compared, then the maximum is found.

# Pseudocode Example 2:
## (in your own language. This is only an example. )

```
FIND MAX (ARRAY X):                                ; program start
  SET MAXIMUM to the first element of X.           ; set initial condition

  WHILE i < X's length THEN                        ; iterate through X array
    BEGIN
        IF X[ i ] > MAXIMUM THEN                   ; update maximum if
            ASSIGN X[i] to MAXIMUM                 ; a greater element found
        INCREMENT i                                ; find next element
    END
  RETURN MAXIMUM                                   ; return maximum
```

# Personal Experience

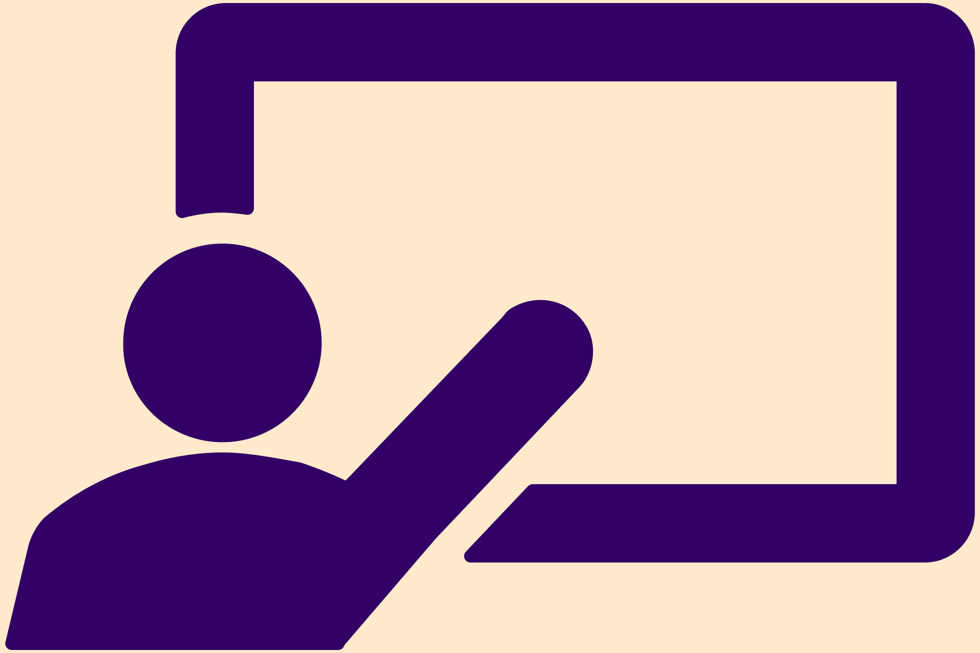
- I personally just use C/C++ or Java-like language as pseudo language to describe a program if it is not coded yet.
- Java is a good candidate for pseudo language as well.

# Homework (Honor code)

Write a pseudo code to perform a task of

**How to prepare breakfast with:**

- **English Muffins**
- **Bacon**
- **A Fried Egg**
- **Orange Juice**
- **Fruit Cup**



# Print Calendar

LECTURE 7

# AUGUST 2023

| SUN | MON | TUE | WED | THU | FRI | SAT |
|---|---|---|---|---|---|---|
| 30 | 31 | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | 31 | 1 | 2 |

# Top Level Analysis

- Input: Month, Year
- Output: Monthly Calendar for the specific month.

# Unit Project: printCalendar.java

Write a program to print out a monthly calendar from any month after Jan. 1800. (Wednesday).

Using the top down design and bottom up implementation guidelines from the previous two lectures.

# Expected Result: (Jan. 2016)

## Sample Answer Program: printCalendar.java

BlueJ: Terminal Window - Unit2

Options

```
Enter full year (e.g., 2016): 2016
Enter month in number between 1 and 12: 1

         January 2016
=============================
 Sun Mon Tue Wed Thu Fri Sat
                       1   2
   3   4   5   6   7   8   9
  10  11  12  13  14  15  16
  17  18  19  20  21  22  23
  24  25  26  27  28  29  30
  31
```