CS 24 AP Computer Science A Review

The Night Before AP Computer Science A Exam

DR. ERIC CHOU
IEEE SENIOR MEMBER

The Night Before AP Computer Science A

eCode24.com

echou510@gmail.com

facebook.com/DrEricChou

eCodeHacker.com

Eric Chou, Ph.D.

IEEE Senior Member
CSTA Member

IEEE

USC

National Taiwan University

Topics

- Multiple Choice Problem Solving Skills
- Free Response Problem Solving Skills

SECTION 1

Updates for 2019-2020

Units in APCSA course

1. Primitive Types
2. Using Objects
3. Boolean Expressions and if Statements
4. Iteration
5. Writing Classes
6. Array
7. ArrayList
8. 2-D Array
9. Inheritance
10. Recursion

Free Responses

The four free-response question types will remain the same from year to year:

- **Question 1:** Methods and Control Structures, where students call methods and work with control structures without the added complexity of data structures.
- **Question 2:** Class, where students design and implement a described class.
- **Question 3:** Array/ArrayList, where students complete program code that uses array or ArrayList objects.
- **Question 4:** 2-D Array, where students complete program code that uses 2-D arrays.

SECTION 2

Unit 1 Elementary Programming

The Basics

- Every AP exam question uses at least one of these:
  - Types and Identifiers
  - Operators
  - Control structures

Identifiers

- Identifiers – name for variable, parameter, constant, user-defined method/class, etc.
  - Convention says identifiers for variables and methods will be lowercase (with uppercase letters to separate multiple words)
  - E.g. `getName, findSurfaceArea, preTaxTotal`
  - Class names will be capitalized
  - E.g. `Student, Car, BankAccount`

Primitive Types

- **primitive types**: 8 simple types for numbers, text, etc. Java also has **Reference Types**, which we'll talk about later

Name	Description	Examples
`int`	integers	`42`, `-3`, `0`, `926394`
`double`	real numbers	`3.1`, `-0.25`, `9.4e3`
`char`	single text characters	`'a'`, `'X'`, `'?'`, `'\n'`
`boolean`	logical values	`true`, `false`

- Why does Java distinguish integers vs. real numbers?

Numeric Data Types and Operations

Java has six numeric types for integers and floating-point numbers with operators +, -, *, . and %

Name	Data	Range	Default Value	Size
byte	signed integer	[-128, 127]	0	8 bits
short	signed integer	[-32768, 32767]	0	16 bits
int	signed integer	[-2147483648, 2147483647]	0	32 bits
long	signed integer	[-9223372036854775808, 9223372036854775807]	0	64 bits
float	floating-point	**MIN:** ±1.4E-45 **MAX:** ±3.4028235E+38	0.0	32 bits
double	floating-point	**MIN:** ±4.9E-324 **MAX:** ±1.79769313486231570E+308	0.0	64 bits
char	Unicode	['\u0000', '\uFFFF']	'\u0000'	16 bits
boolean	logical value	{false, true}	false	≥ 1 bit

Type casting

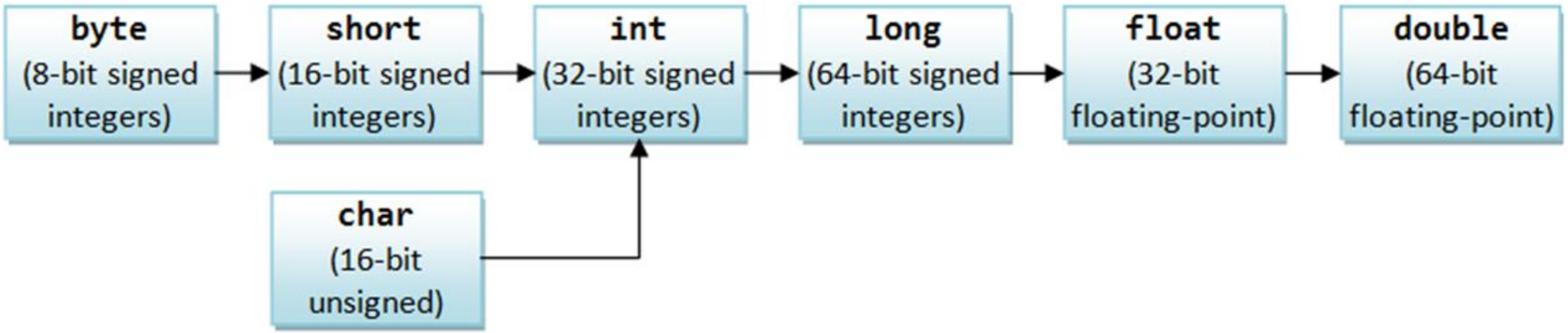
- **type cast**: A conversion from one type to another.
  - To promote an int into a double to get exact division from /
  - To truncate a double from a real number to an integer

- Syntax:

  (**type**) **expression**

  Examples:

```
double result = (double) 19 / 5;        // 3.8
int result2 = (int) result;             // 3
int x = (int) Math.pow(10, 3);          // 1000
```



Orders of Implicit Type-Casting for Primitives

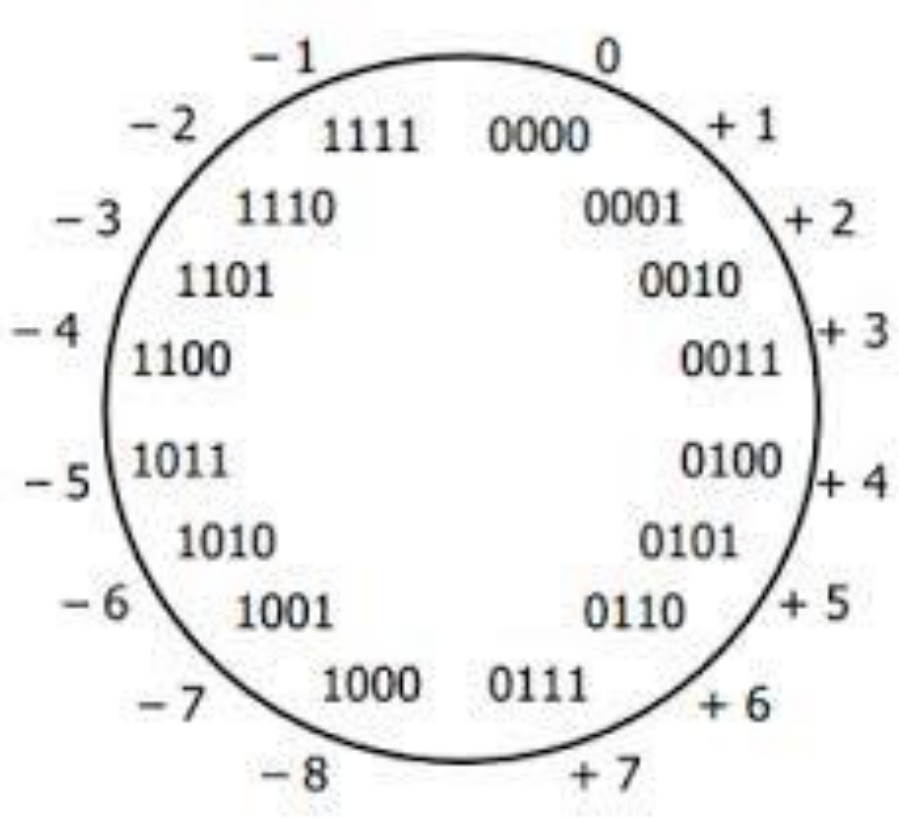
Storage of numbers

- int types use 32 bits, largest integer is 2^{31} - 1
- Floating-point numbers use mantissa and exponent: sign*mantissa*$2^{exponent}$
- When floating point numbers are converted to binary, most cannot be represented exactly, leading to **round-off error**

Integer.MIN_VALUE and Integer.MAX_VALUE

- Represent the absolute lowest and highest values that can be stored in an integer
- If you're trying to find the minimum or maximum value of an array or ArrayList, initialize your variable to these

Two's Complement



Negative number is represented as two's complement.

For byte number's (8 bits):

$-X = (2^8 - 1) - X + 1;$

$X + (-X) = X + (2^8 - X) = 2^8 = 0\ ;$

eg.

A = 0100 -> A's One's Complement = 1011 ->

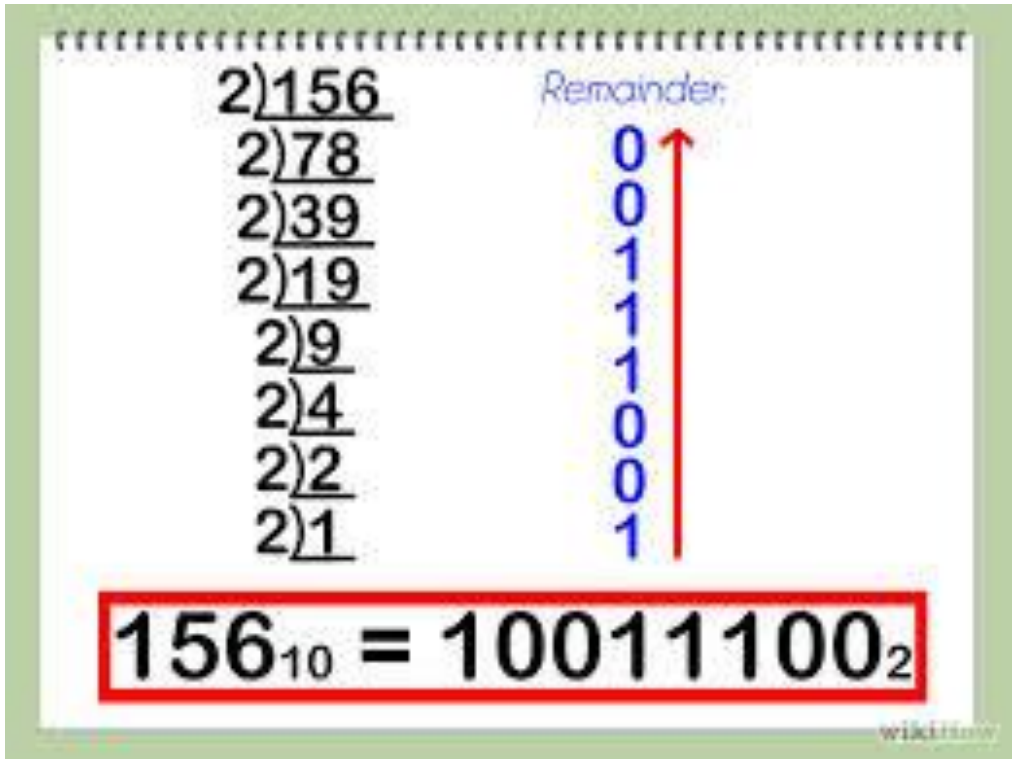
A's Two's Complement -> 1100

The number 2^8 is a overflow for the byte format, because unsigned byte number range

from 0 to $2^8 - 1$ = 11111111.

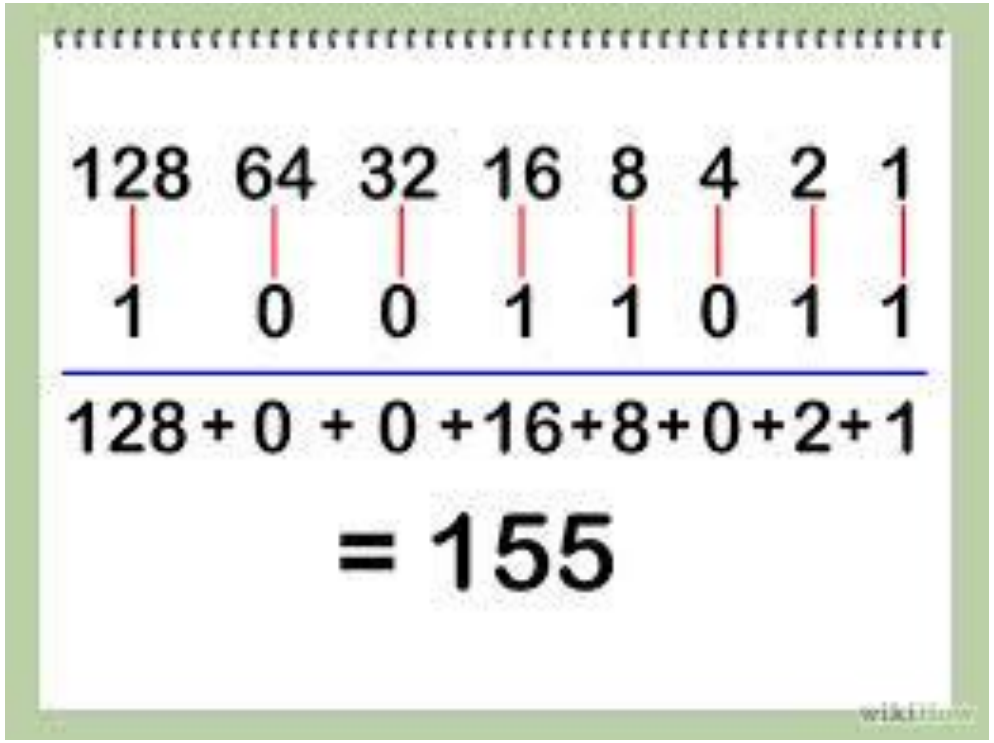
Therefore, this method can work for computer.

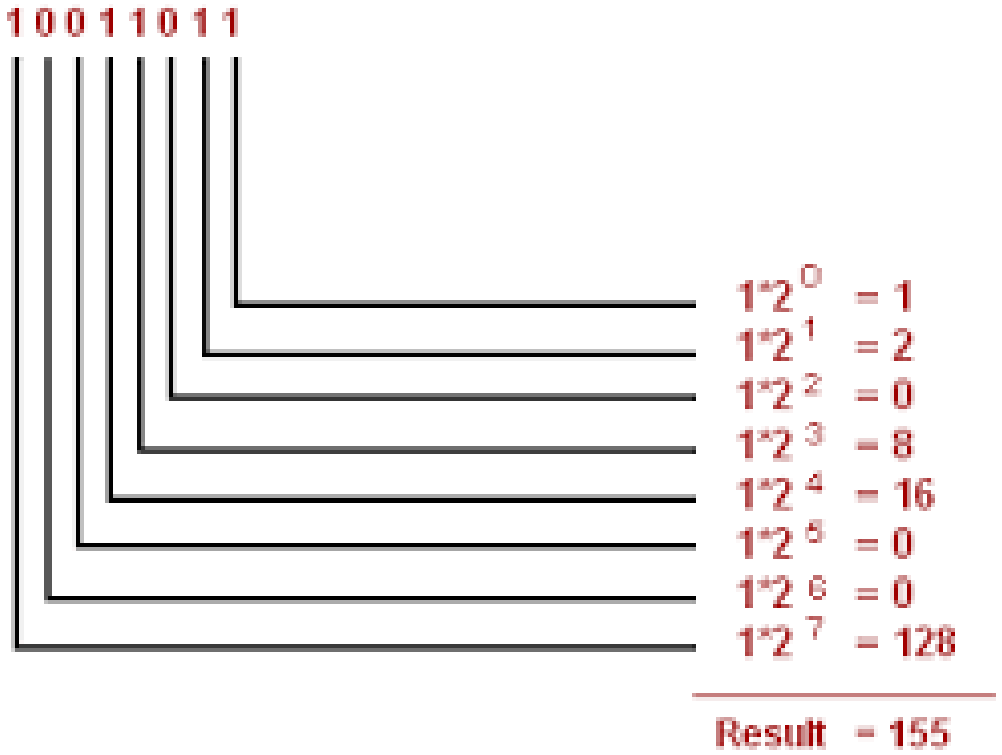
Decimal to Binary



Divider	Dividend	Remainder
2	202	0
2	101	1
2	50	0
2	25	1
2	12	0
2	6	0
2	3	1
		1

Binary/Decimal Conversion





Constant Variables

final variable is a quantity whose value will not change

E.g. final int CLASS_SIZE = 30

Arithmetic Operators

Operator	Meaning	Example
+	Addition	3 + x
-	Subtraction	p – q
*	Multiplication	6 * i
/	Division	10 / 4 //returns 2
%	Mod (remainder)	11 % 8 //returns 3

Arithmetic Operators Notes

- Integer division truncates the answer (cuts off the decimal)
- Use type casting to control how to divide.
- Which do not evaluate to 0.75?
  - 3.0 / 4
  - 3 / 4.0
  - (int) 3.0 / 4
  - (double) 3 / 4
  - (double) (3 / 4)

Boolean Data Values in Java

- Boolean Value: true/false
- Boolean Variable: boolean b = a<3;
- Boolean Expression: (a + b) < (c + d)
- Boolean Function:

```
boolean f(int x) {   return x % 2 == 0; }
```

Type boolean

- **boolean**: A logical type whose values are true and false.
  - A **test** in an if, for, or while is a boolean expression.
  - You can create boolean variables, pass boolean parameters, return boolean values from methods, ...

```
boolean minor = (age < 21);
boolean expensive = iPhonePrice > 200.00;
boolean iLoveCS = true;
if (minor) {
    System.out.println("Can't purchase alcohol!");
}
if (iLoveCS || !expensive) {
    System.out.println("Buying an iPhone");
}
```

De Morgan's Law

- **De Morgan's Law**:
  Rules used to *negate* or *reverse* boolean expressions.
  - Useful when you want the opposite of a known boolean test.

Original Expression	Negated Expression	Alternative
a && b	!a || !b	!(a && b)
a || b	!a && !b	!(a || b)

  - Example:

Original Code	Negated Code

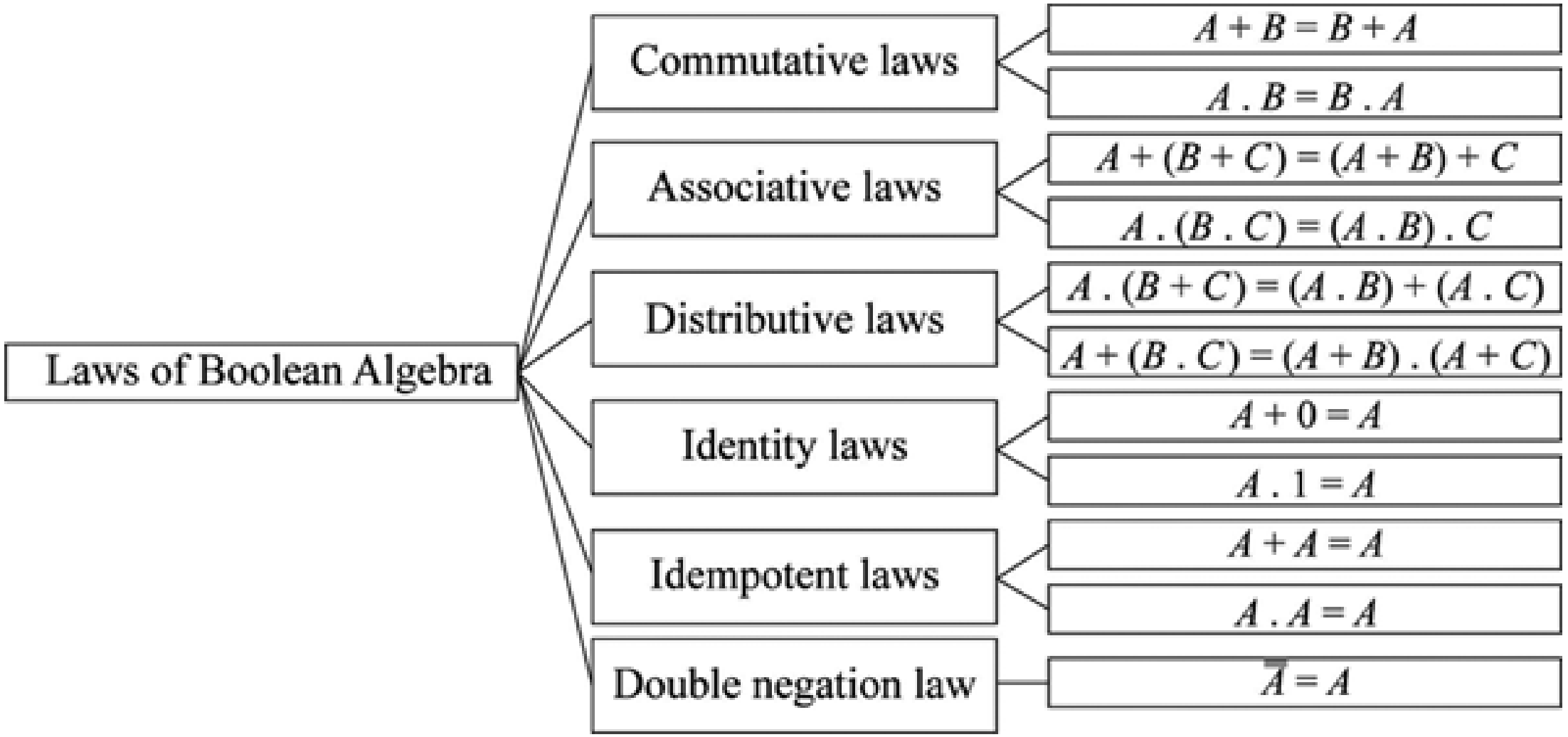
if (x == 7 && y > 3) {
  ...
}
	if (x != 7 || y <= 3) {
	  ...
	}

Relational Operators

Operator	Meaning	Example
==	Equal to	if (x == 100)
!=	Not equal to	if (age != 21)
>	Greater than	if (salary > 30000)
<	Less than	if (grade < 65)
>=	Greater than or equal to	if (age >= 16)
<=	Less than or equal to	if (height <= 6)

Logical Operators

Operator	Meaning	Example
!	NOT	if (!found)
&&	AND	if (x < 3 && y > 4)
||	OR	if (age < 2 || height < 4)


Laws of Boolean Algebra
Commutative laws
$A + B = B + A$
$A \cdot B = B \cdot A$
Associative laws
$A + (B + C) = (A + B) + C$
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$
Distributive laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
$A + (B \cdot C) = (A + B) \cdot (A + C)$
Identity laws
$A + 0 = A$
$A \cdot 1 = A$
Idempotent laws
$A + A = A$
$A \cdot A = A$
Double negation law
$\overline{\overline{A}} = A$

Law/Theorem	Law of Addition	Law of Multiplication
Identity Law	$x + 0 = x$	$x \cdot 1 = x$
Complement Law	$x + x' = 1$	$x \cdot x' = 0$
Idempotent Law	$x + x = x$	$x \cdot x = x$
Dominant Law	$x + 1 = 1$	$x \cdot 0 = 0$
Involution Law	$(x')' = x$	
Commutative Law	$x + y = y + x$	$x \cdot y = y \cdot x$
Associative Law	$x+(y+z) = (x+y)+z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
Distributive Law	$x \cdot (y+z) = x \cdot y+x \cdot z$	$x+y \cdot z = (x+y) \cdot (x+z)$
Demorgan's Law	$(x+y)' = x' \cdot y'$	$(x \cdot y)' = x' + y'$
Absorption Law	$x + (x \cdot y) = x$	$x \cdot (x + y) = x$

$$A\bar{B}\bar{C} + \bar{A}BC$$

$2^n = 2^3 = 8/2 = 4/2$

$= 2/2 = 1$

A	B	C	$\bar{A}$	$\bar{B}$	$\bar{C}$	$A\bar{B}\bar{C}$	$\bar{A}BC$	$A\bar{B}\bar{C} + \bar{A}BC$
0	0	0	1	1	1			
0	0	1	1	1	0			
0	1	0	1	0	1			
0	1	1	1	0	0			
1	0	0	0	1	1			
1	0	1	0	1	0			
1	1	0	0	0	1			
1	1	1	0	0	0			

2^n

AND

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

Logical Operators Example

(x && y) || !(x && y)

A. Always true
B. Always false
C. true only when x is true and y is true
D. true only when x and y have the same value
E. true only when x and y have different values

Another example

Which is equivalent to:
!(a < b) && !(a > b)

A. true
B. false
C. a == b
D. a != b
E. !(a < b) && (a > b)

Assignment Operators

Operator	Example	Meaning
=	x = 2	Simple assignment
+=	x += 4	x = x + 4
-=	y -= 6	y = y – 6
*=	p *= 5	p = p * 5
/=	n /= 10	n = n / 10
%=	n %= 10	n = n % 10
++	k++	k = k + 1
--	i--	i = i - 1

Operators	Notation	Precedence/Priority
Postfix	expr++ , expr--	1
Unary	++expr , --expr , +expr –expr , ~ , !	2
Multiplicative	* , / , %	3
Additive	+ , -	4
Shift	<< , >> , >>>	5
Relational	< , > , <= , >= , instanceof	6
Equality	== , !=	7
Bitwise AND	&	8
Bitwise Exclusive OR	^	9
Bitwise Inclusive OR	|	10
Logical AND	&&	11
Logical OR	||	12
Ternary	? :	13
Assignment	= , += , -= , *= , /= , %= , &= , ^= , |= , <<= , >>= , >>>=	14

ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Class Character

class Character contains useful methods

- Examples of useful `Character` methods:
  - `Character.isDigit(c)`
  - `Character.isLetter(c)`
  - `Character.isWhitespace(c)`
  - `Character.isLowerCase(c)`
  - `Character.toLowerCase(c)`
  - see Java API for more!
- These methods are `static` and are applied to `char c`

Character Methods

Method	Description
`isUpperCase()`	Tests if character is uppercase
`toUpperCase()`	Returns the uppercase equivalent of the argument; no change is made if the argument is not a lowercase letter
`isLowerCase()`	Tests if character is lowercase
`toLowerCase()`	Returns the lowercase equivalent of the argument; no change is made if the argument is not an uppercase letter
`isDigit()`	Returns `true` if the argument is a digit (0–9) and `false` otherwise
`isLetter()`	Returns `true` if the argument is a letter and `false` otherwise
`isLetterOrDigit()`	Returns `true` if the argument is a letter or digit and `false` otherwise
`isWhitespace()`	Returns `true` if the argument is whitespace and `false` otherwise; this includes the space, tab, newline, carriage return, and form feed

Java's `Math` class

Method name	Description
`Math.abs(`*value*`)`	absolute value
`Math.ceil(`*value*`)`	rounds up
`Math.floor(`*value*`)`	rounds down
`Math.log10(`*value*`)`	logarithm, base 10
`Math.max(`*value1*`, `*value2*`)`	larger of two values
`Math.min(`*value1*`, `*value2*`)`	smaller of two values
`Math.pow(`*base*`, `*exp*`)`	*base* to the *exp* power
`Math.random()`	random `double` between 0 and 1
`Math.round(`*value*`)`	nearest whole number
`Math.sqrt(`*value*`)`	square root

`Math.sin(`*value*`)`
`Math.cos(`*value*`)`
`Math.tan(`*value*`)`
	sine/cosine/tangent of an angle in radians

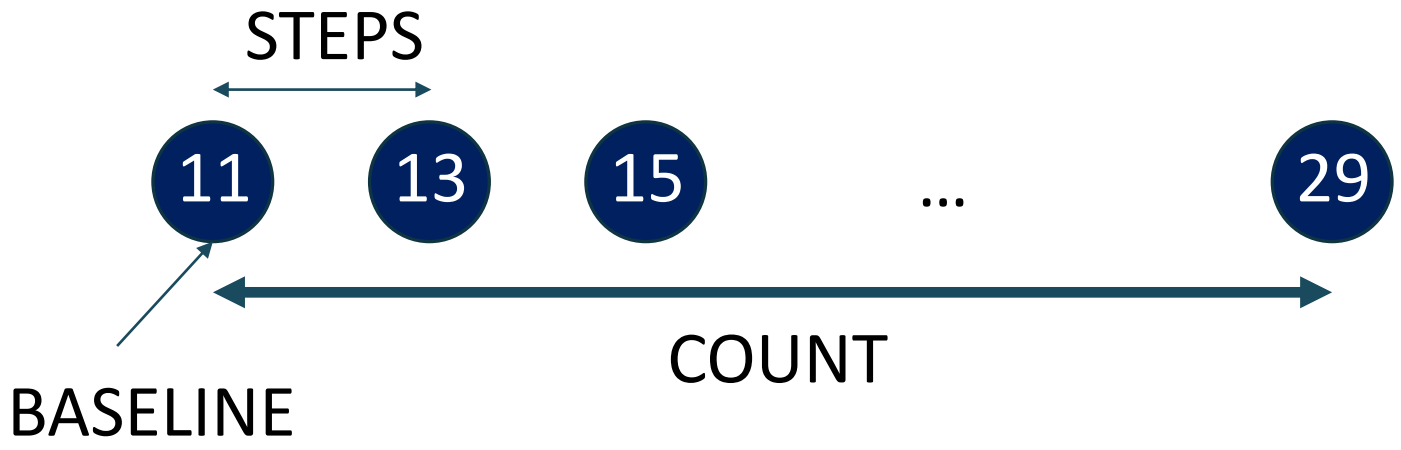
`Math.toDegrees(`*value*`)` `Math.toRadians(`*value*`)`	convert degrees to radians and back

Constant	Description
`Math.E`	2.7182818...
`Math.PI`	3.1415926...

Review of Random Sample Generation

int r = ((int) (Math.random()*COUNT) * STEPS + BASELINE;

int r = ((int) (Math.random() * 10) * 2 + 11);



Strings

- **string** : An object storing a sequence of text characters.

  String **name** = "**text**";

  String **name** = **expression**;

  - Characters of a string are numbered with 0-based *indexes* :

    String name = "P. Diddy";

index	0	1	2	3	4	5	6	7
char	P	.		D	i	d	d	y

  - The first character's index is always 0
  - The last character's index is 1 less than the string's length
  - The individual characters are values of type char

String methods

Method name	Description
indexOf(**str**)	index where the start of the given string appears in this string (-1 if it is not there)
length()	number of characters in this string
substring(**index1**, **index2**) or substring(**index1**)	the characters in this string from *index1* (inclusive) to *index2* (exclusive); if *index2* omitted, grabs till end of string
compareTo(String other) returns <0 if this is less than other	returns 0 if this is equal to other returns >0 if this is greater than other

- These methods are called using the dot notation:

```
String gangsta = "Dr. Dre";
System.out.println(gangsta.length());    // 7
```

Note: These are the only String methods required for the AP CS...

The equals method

- Objects are compared using a method named equals.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- Technicallythis is a method that returns a value of type boolean, the type used in logical tests.

Java String Methods

(1) length()
(2) charAt()
(3) trim()
(4) indexOf()
(5) lastIndexOf()
(6) replace()
(7) toLowerCase()
(8) toUpperCase()
(9) split()
(10) substring()
(11) equals()
(12) getBytes()
(13) concat()
(14) contains()
(15) compareTo
(16) startsWith()
(17) endsWith()
.
.
.
etc.

SECTION 3

Unit 2 Structured Programming

Control Structures

if

if…else

if…else if

while loop

for loop

for-each loop

if example

- 3 bonus questions → must get all correct for 5 points added to **`grade`**
- **`bonus1`**, **`bonus2`**, and **`bonus3`** are `boolean` variables that indicate whether they are correct
- Write an if statement for this example
  - If (bonus1 && bonus2 && bonus3)
    grade += 5;

if...else Statement

```
if (boolean expression)
{
   statements
   //will be executed if boolean expression is true
}
else
{
   statements
   //will be executed if boolean expression is false
}
```

if...else if

```
if (grade.equals(“A”))
    System.out.println(“Excellent”);
else if(grade.equals(“B”))
    System.out.println(“Good”);
else if(grade.equals(“C”))
    System.out.println(“Poor”);
else
    System.out.println(“Invalid”);
```

Combination of if-statements

if/if…else if statements do not always need an else at the end

An if statement inside of an if statement is a nested if statement:

```
if (boolean expr1)
    if (boolean expr2)
        statement;
```

Can also be written as:

```
if (boolean expr1 && boolean expr2)
    statement;
```

Rewrite using only if…else

```
if(value < 0)
    return “Not in range”;
else if(value > 100)
    return “Not in range”;
else
    return “In range”;
```

```
if(value < 0 || value > 100)
    return “Not in range”;
else
    return “In range”;
```

Dangling if

The **else** belongs to the closest if-statement:

```
if (a > 3 )
    if (b > 4) c++;
else  d++;
```

While vs. For loops

WHILE LOOPS

```
int i = 0;
while (i < 100)
{
        //repeated code
        i++;
}
```

FOR LOOPS

```
for(int i = 0; i<100; i++)
{
        //repeated code
}
```

While vs. For-each loops

WHILE LOOP

```
int[] locationCells
                    = new int[100];
int i = 0;
while (i < 100)
{
          System.out.println
               (locationCells[i]);
          i++;
}
```

FOR-EACH LOOP

```
int[] locationCells
                    = new int[100]; for(int
cell : locationCells)
{
   System.out.println(cell);
}
```

For loop vs. for-each loop

FOR LOOP

```
for(int i = 0; i<100; i++)
{
        System.out.println
            (locationCells[i]);
}
```

FOR-EACH LOOP

```
for(int cell : locationCells)
{
   System.out.println(cell);
}
```

For loop vs. for-each loop

FOR LOOP	FOR-EACH LOOP
Has an index → useful for setting data that depends on the index	Easier to write when simply accessing data from an array
Initialization, boolean test, and iteration expression are all in one line	Not much better than a while loop if not accessing array data

While loop example

```
int value = 15;
while (value < 28) {
    System.out.println(value);
    value++;
}
```

What is the first number printed?

- 15

What is the last number printed?

- 27

Another example

```
int a = 24;
int b = 30;
while (b != 0) {
    int r = a % b;
    a = b;
    b = r;
}
System.out.println(a);
```

A. 0

B. 6

C. 12

D. 24

E. 30

Yet another example

```
int k = 0;
while (k < 10) {
    System.out.print((k % 3) + “  ”);
    if ((k % 3) == 0)
        k = k + 2;
    else
        k++;
}
```

A. 0 2 1 0 2

B. 0 2 0 2 0 2

C. 0 2 1 0 2 1 0

D. 0 2 0 2 0 2 0

E. 0 1 2 1 2 1 2

For loop example

```
String str = “abcdef”;
for (int r = 0; r < str.length()-1; r++)
   System.out.print(str.substring(r, r+2));
```

What is printed?

A. abcdef
B. aabbccddeeff
C. abbccddeef
D. abcbcdcdedef
E. Nothing, IndexOutOfBoundsException thrown

Yet another example

```
for (int outer = 0; outer < n; outer++)
    for(int inner = 0; inner <= outer; inner++)
        System.out.print(outer + “  ”);
```

If n has a value of 4, what is printed?

A. 0 1 2 3

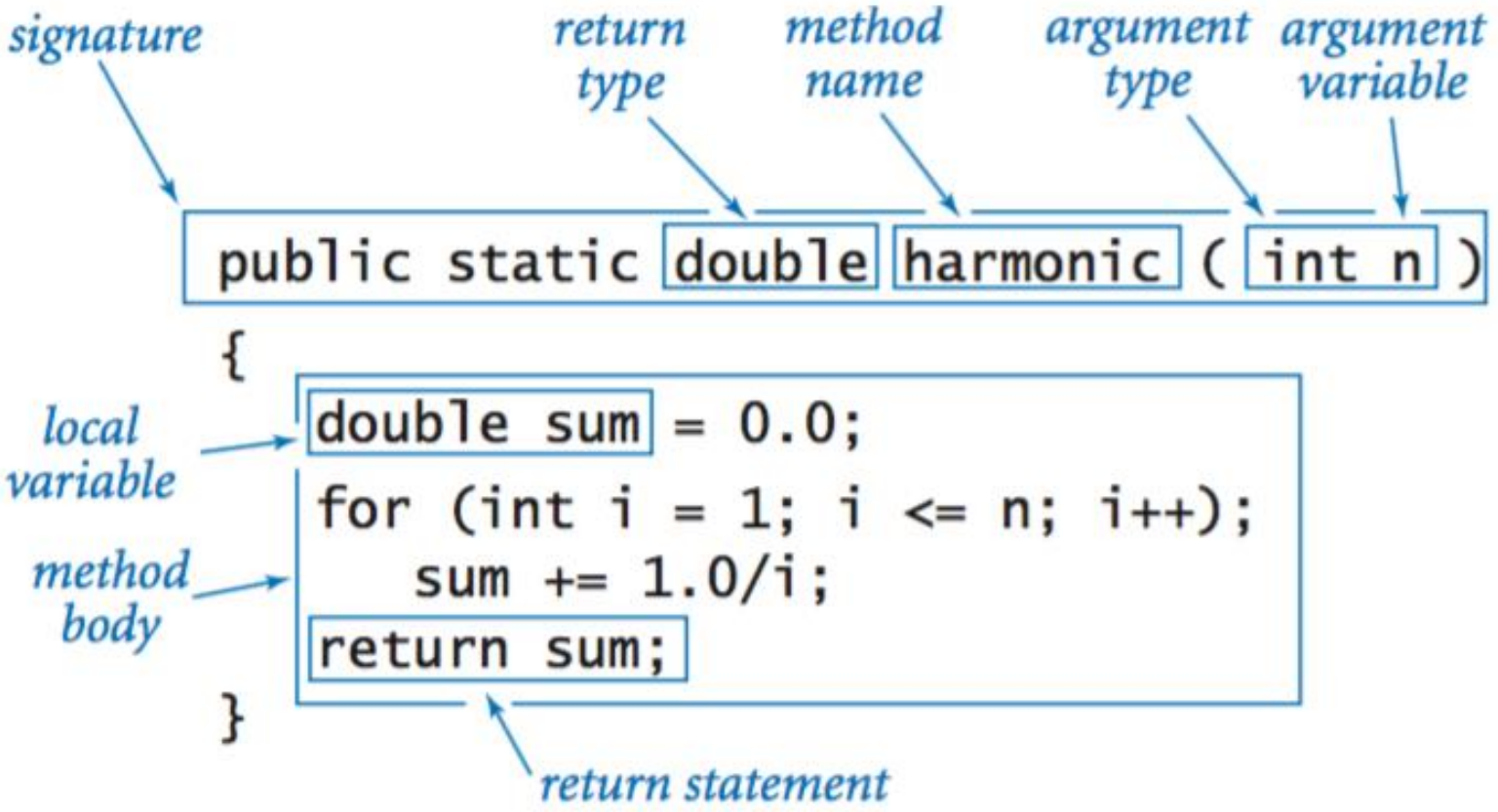
B. 0 0 1 0 1 2

C. 0 1 2 2 3 3 3

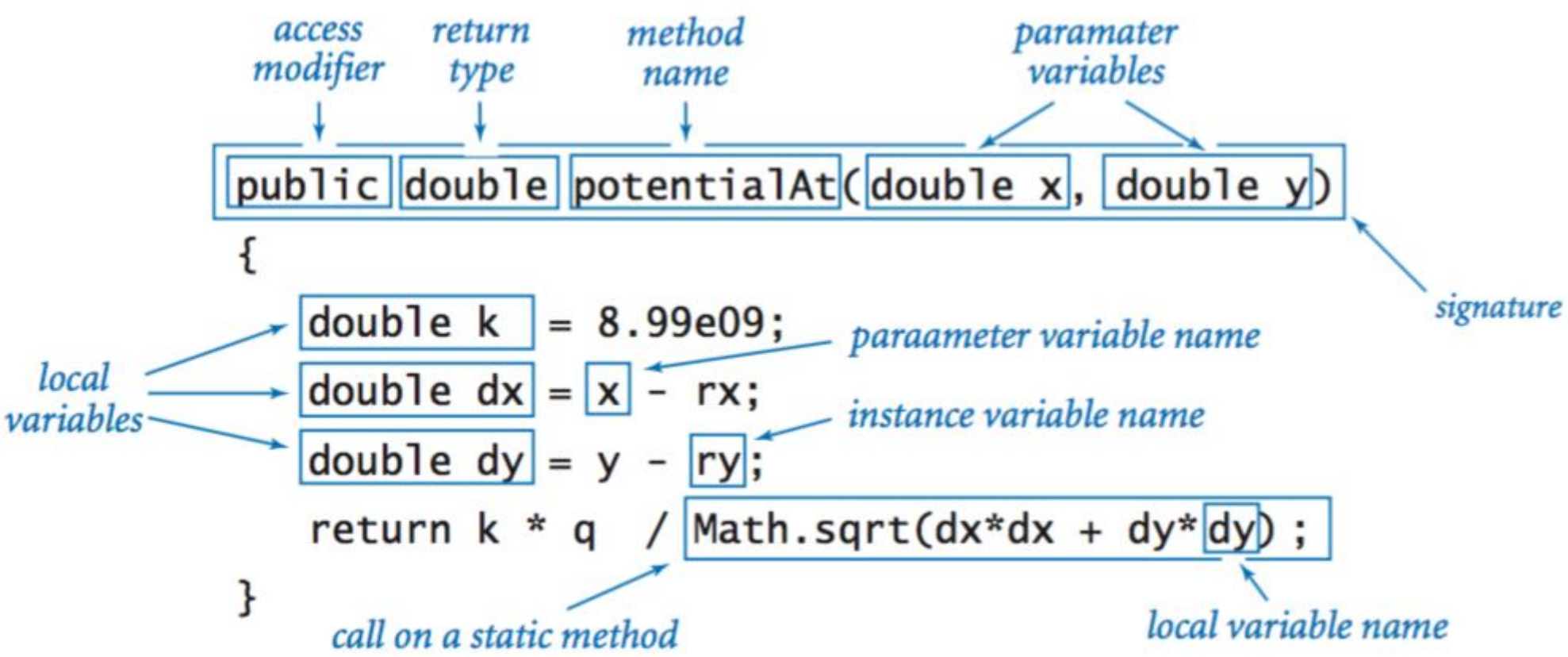
D. 0 1 1 2 2 2 3 3 3 3

E. 0 0 1 0 1 2 0 1 2 3

Method

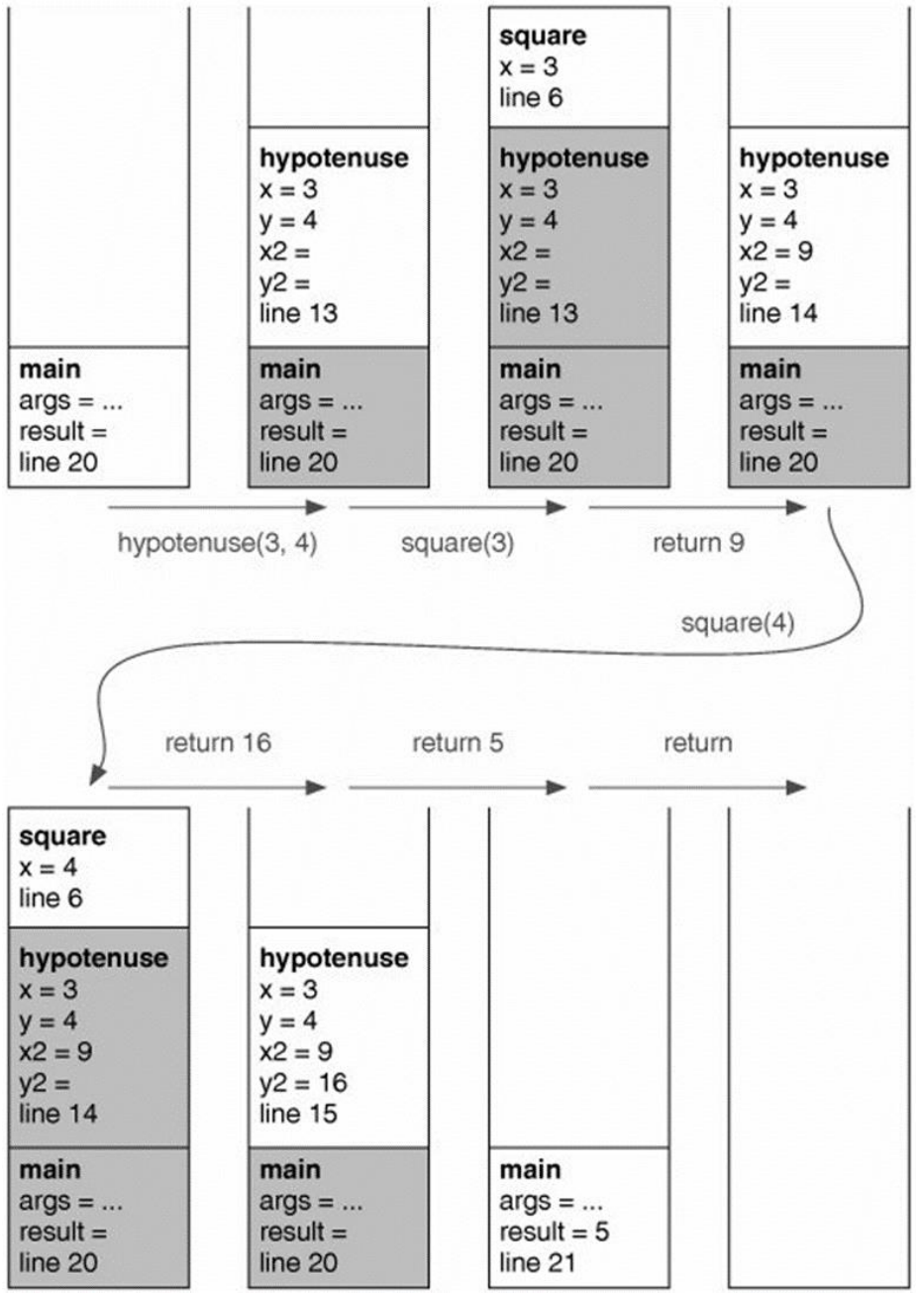

signature
return type
method name
argument type
argument variable
public static double harmonic (int n)
{
local variable
double sum = 0.0;
method body
for (int i = 1; i <= n; i++);
sum += 1.0/i;
return sum;
}
return statement

Instance Method

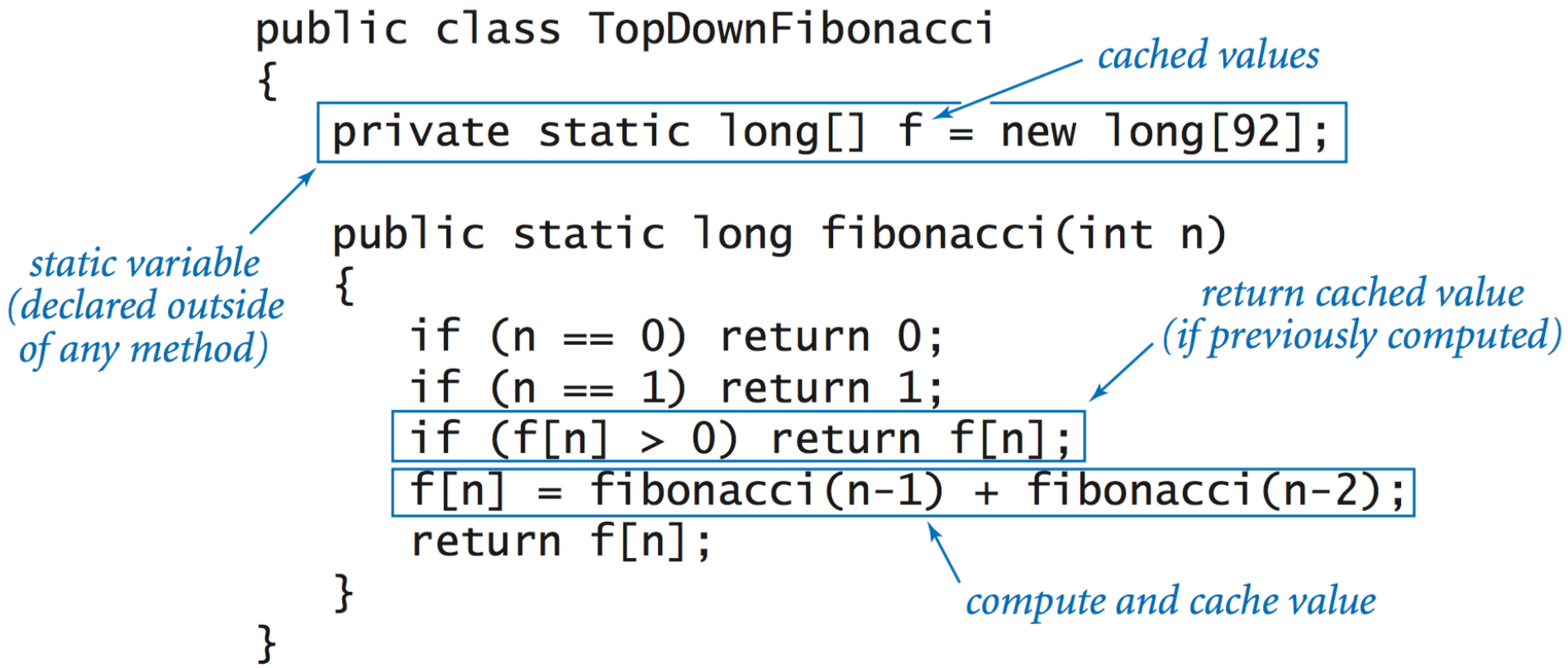


Call Stack

```
1 /** Compute the hypotenuse of a right triangle. */
2 public class Hypotenuse {
3         /** Return the square of the number x. */
4         public static double square(double x) {
5         return x * x;
6     }
7     /**
8     * Return the hypotenuse of a right triangle with side lengths x and y.
9     */
10    public static double hypotenuse(double x, double y) {
11        double x2 = square(x);
12        double y2 = square(y);
13        return Math.sqrt(x2 + y2);
14    }
15    /** Test the methods. */
16    public static void main(String[] args) {
17        double result = hypotenuse(3, 4);
18        System.out.println(result);
19    }
20 }
```



Recursive Method



Method Overloading

- **method overloading**: The ability to define two different or more different methods with the same name but different number and/or type of parameters.

```
public static void drawBox() { // no parameters
  // has code that creates a standard sized box
  ...
}
public static void drawBox(int height, int width) {
  // code that draws the box based on the height and
  // width parameter values
  ...
}
```

Which method used is based on how it is called:

```
drawBox();   // uses the first drawBox method
drawBox(10,20);   // uses second drawBox method
```

SECTION 4

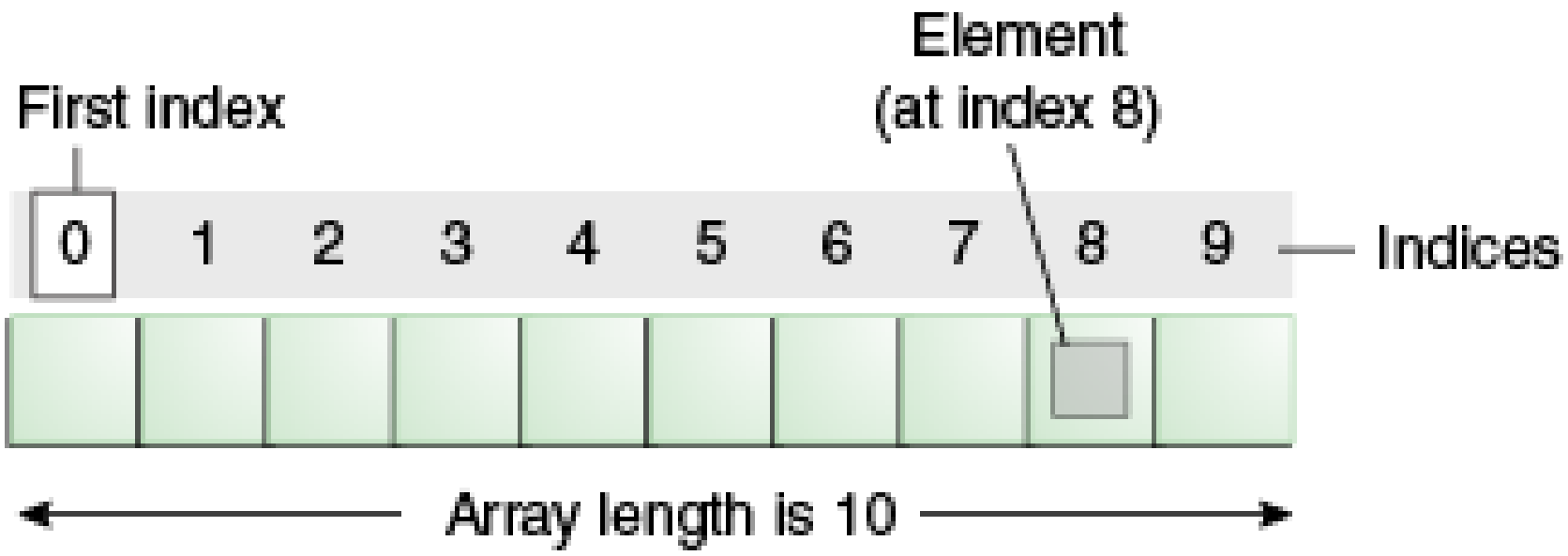
Unit 3
Data Structures

Lists and Arrays

Manipulate a list. Search, delete and insert an item. Very common on the AP exam.

- One-dimensional arrays
- ArrayLists
- Two-dimensional arrays

1-D Arrays

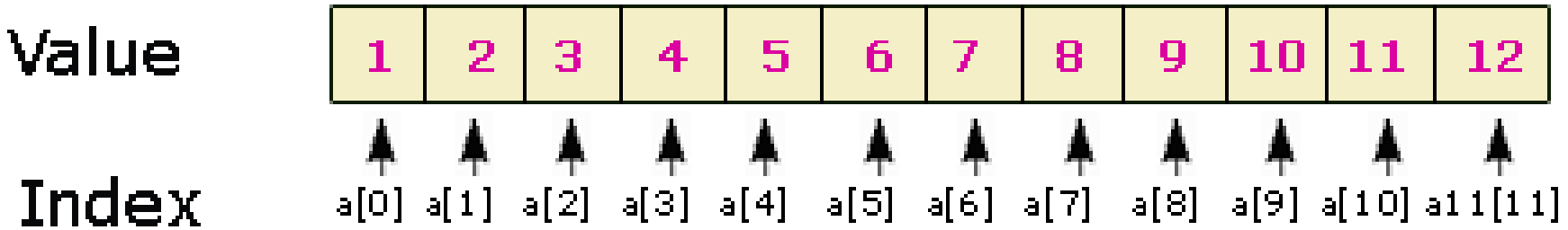

Element
(at index 8)
First index
0
1
2
3
4
5
6
7
8
9
Indices
Array length is 10

1-D Array Initialization

- Which of these are valid ways to assign a reference to an array?
  - double[] data = new double[25];
  - double data[] = new double[25];
  - double[] data;
    data = new double[25];
- All are three are valid!

One Dimensional array

Initialization `int a[] = new int [12];`




```
System.out.print(a[5]);
```

Output: 6

Array Length

- length is a public instance variable of arrays:

```
String[] names = new String[25];
names.length;                    //returns 25
```

- Array indices go from 0 to names.length-1 (i.e. 0 to 24)
- length is not a method for arrays; length is a method for Strings

Traversing an Array

- Use for-each loop when you need to access (only access) every element in an array without replacing or removing elements
- Use for loop for all other cases

What to do with arrays

You need to be able to read and write code that accomplishes each of the following:

- Counting elements
- Printing elements
- Summing elements
- Swapping elements
- Finding the minimum or maximum
- Inserting elements
- Deleting elements

Counting & Printing

Counting:

```
int total = 0;
for(int i = 0; i<arr.length; i++) {
    total++;
}
```

Printing:

```
for(int i = 0; i<arr.length; i++) {
    System.out.println(arr[i]);
}
```

Summing Values

The method calcTotal is intended to return the sum of all values in vals.

```
private int[] vals;
public int calcTotal() {
    int total = 0;
    /* missing code */
    return total;
}
```

What code should replace /* missing code */ in order for calcTotal to work correctly?

Summing Values

```
private int[] vals;
public int calcTotal() {
    int total = 0;
    for(int pos = 0; pos < vals.length; pos++) {
        total += vals[pos];
    }
    return total;
}
```

Summing Values

```
private int[] vals;
public int calcTotal() {
    int total = 0;
    int pos = 0;
    while (pos < vals.length) {
        total += vals[pos];
        pos++;
    }
    return total;
}
```

Swapping values

int[] arr = new int[10];

How to swap arr[0] and arr[5]?

A.
```
arr[0] = 5;
arr[5] = 0;
```

B.
```
arr[0] = arr[5];
arr[5] = arr[0];
```

C.
```
int k = arr[5];
arr[0] = arr[5];
arr[5] = k
```

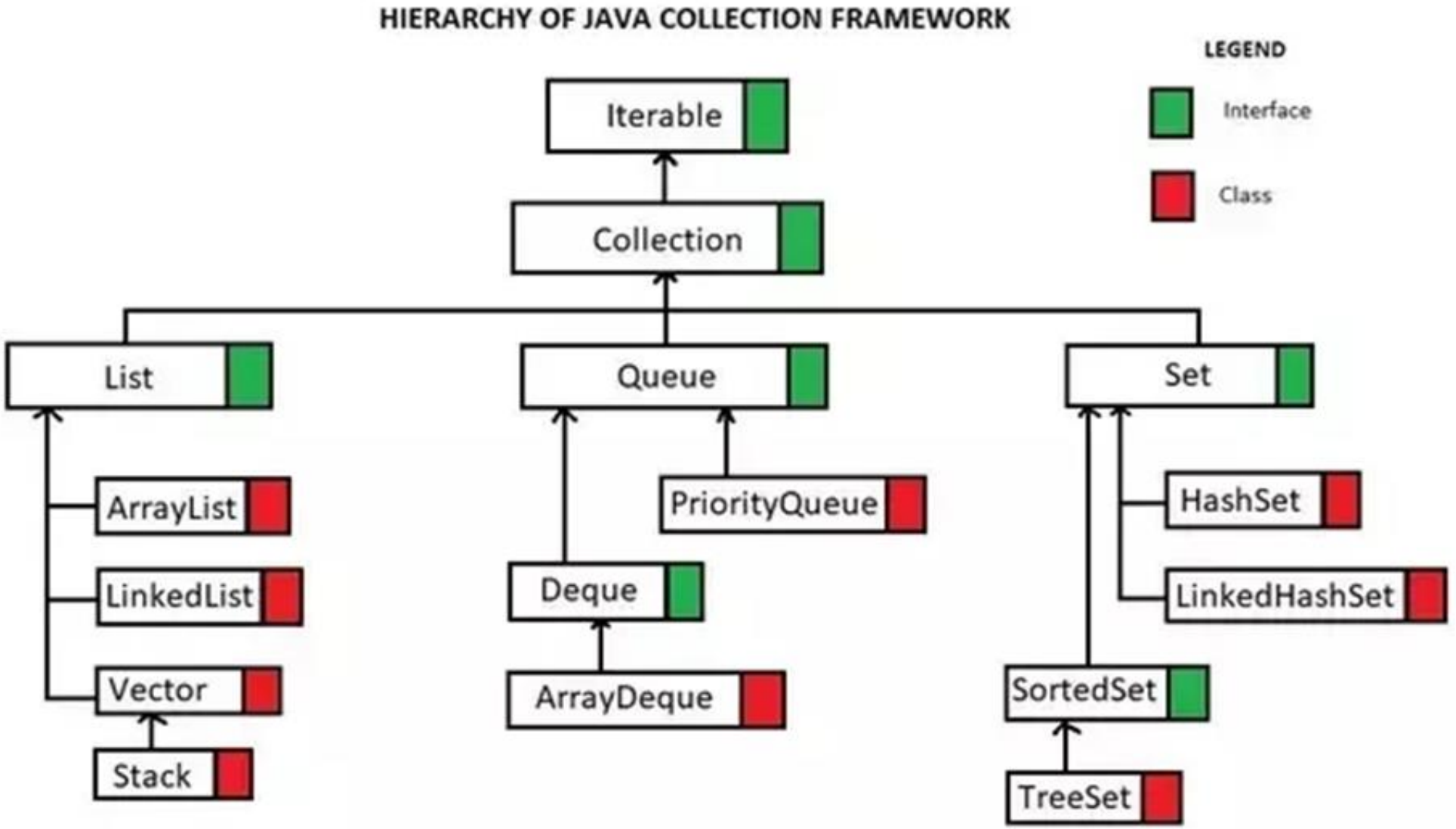
D.
```
int k = arr[0];
arr[0] = arr[5];
arr[5] = k;
```

E.
```
int k = arr[5];
arr[5] = arr[0];
arr[0] = arr[5];
```

Min and Max

```
int min = arr[0];
for(int j = 0; j<arr.length; j++){
    if (arr[j]<min)
        min = arr[j];
}

int max = arr[0];
for(int j = 0; j<arr.length; j++){
    if (arr[j]>max)
        max = arr[j];
}
```


HIERARCHY OF JAVA COLLECTION FRAMEWORK
LEGEND
Interface
Class
Iterable
Collection
List
Queue
Set
ArrayList
LinkedList
Vector
Stack
PriorityQueue
Deque
ArrayDeque
HashSet
LinkedHashSet
SortedSet
TreeSet

ArrayList

- boolean add(Object e)
- void add(int index, Object element)
- boolean addAll(Collection c)

- Object get(int index)
- Object set(int index,Object element)

- Object remove(int index)

- Iterator iterator()
- ListIterator listIterator()

- int indexOf()
- int lastIndexOf()

- int index(Object element)
- int size()
- void clear()

Arrays vs. ArrayList

ARRAYS

```
String[] arr = new String[10];
…
//insert Strings into array
…
for(int i=0; i<arr.length; i++)
{
   System.out.println(arr[i]);
}
```

ARRAYLIST

```
ArrayList<String> arrList = new
ArrayList<String>();
…
//insert Strings into ArrayList
…
for(int i=0; i<arr.size(); i++)
{
   System.out.println
         (arrList.get(i));
}
```

Arrays vs. ArrayList

ARRAYS

```
String[] arr = new String[10];
…
//insert Strings into array
…
for(String x : arr)
{
   System.out.println(x);
}
```

ARRAYLIST

```
ArrayList<String> arrList = new
ArrayList<String>();
…
//insert Strings into ArrayList
…
for(String x : arrList)
{
 System.out.println(x);
}
```

Arrays vs. ArrayList

ARRAYS

Fixed length, set when it is created

Must keep track of last slot if array is not full

Must write code to shift elements if you want to insert or delete

ARRAYLIST

Shrinks and grows as needed

Last slot is always arrList.size()-1

Insert with just arrList.add(object)
Delete with just arrList.remove(objectIndex) or arrList.remove(object)

Insert and Delete

If asked to insert or delete for arrays, you'll likely need to create a new array

More likely asked about ArrayLists

- ArrayLists can change length more easily

ArrayList Question

```
ArrayList<String> items =
        new ArrayList<String>();
items.add("A");
items.add("B");
items.add("C");
items.add(0, "D");
items.remove(3);
items.add(0, "E");
System.out.println(items);
```

A. [A, B, C, E]

B. [A, B, D, E]

C. [E, D, A, B]

D. [E, D, A, C]

E. [E, D, C, B]

Another ArrayList Question

```
public void replace(ArrayList<String> nameList,
                                String name, String newValue) {
   for (int j = 0; j<nameList.size(); j++) {
      if( /* expression */ )
         nameList.set(j, newValue);
   }
}
```

What should be used to replace **/*expression*/** so that the **replace** method will replace all occurrences of **name** in **nameList** with **newValue**?

nameList.get(j).equals(name)

removeAll

- Write the removeAll method that will remove all instances of String str from ArrayList arrList and return the number of items removed

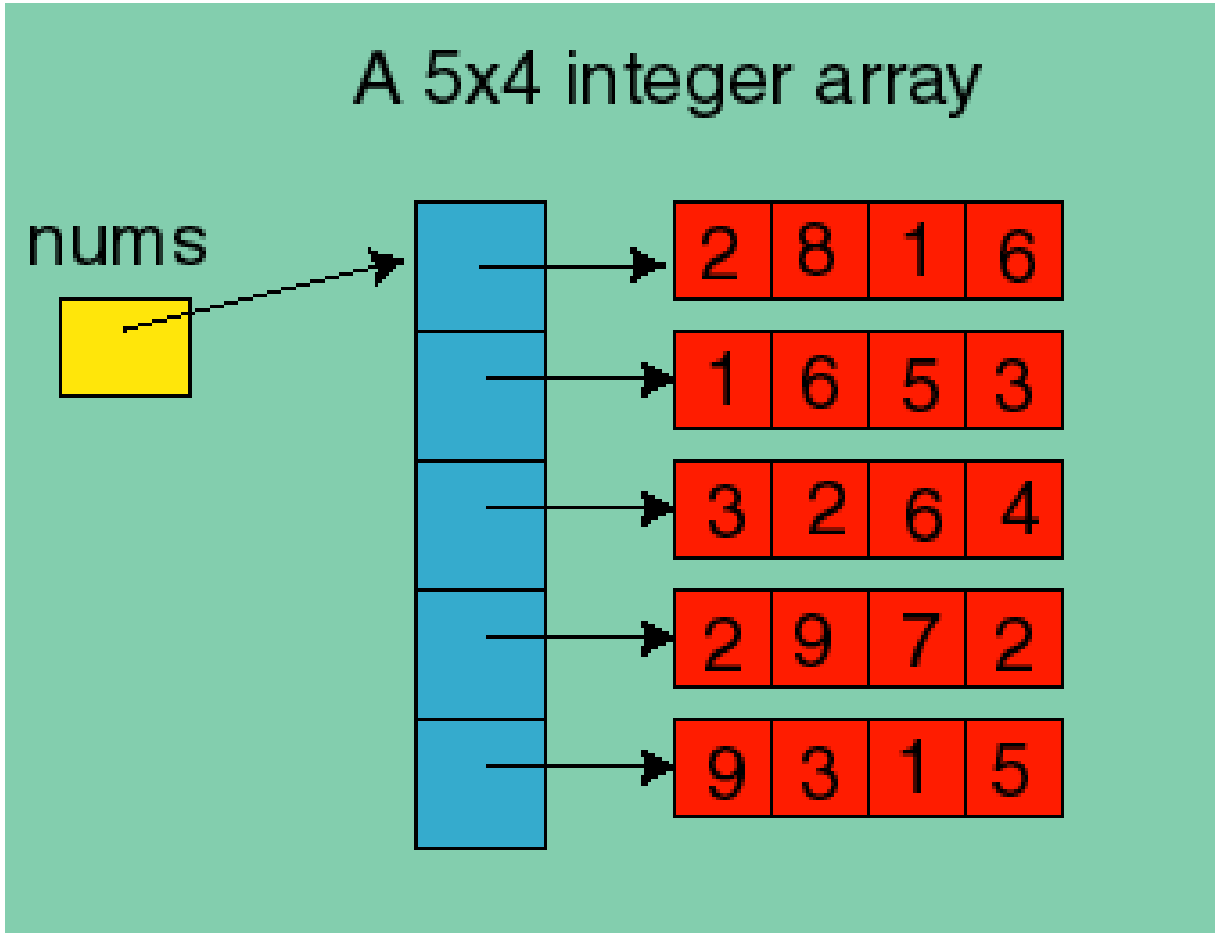
```
public int removeAll(ArrayList<String> arrList,
                     String str) {
   /* complete this method */
}
```

removeAll

```
public int removeAll(ArrayList<String> arrList, String str) {
    int numRemoved = 0;
    for (int i = arrList.size()-1; i>=0; i--) {
        if(str.equals(arrList.get(i)) {
            numRemoved += 1;
            arrList.remove(i);
        }
    }
}
```

2-D Arrays

```
int[][] nums = new int[5][4];
```



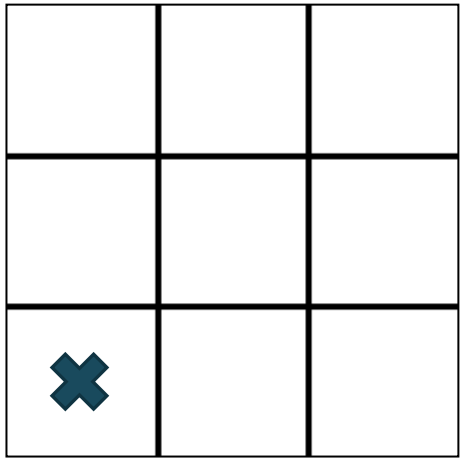
2-D Array as a table

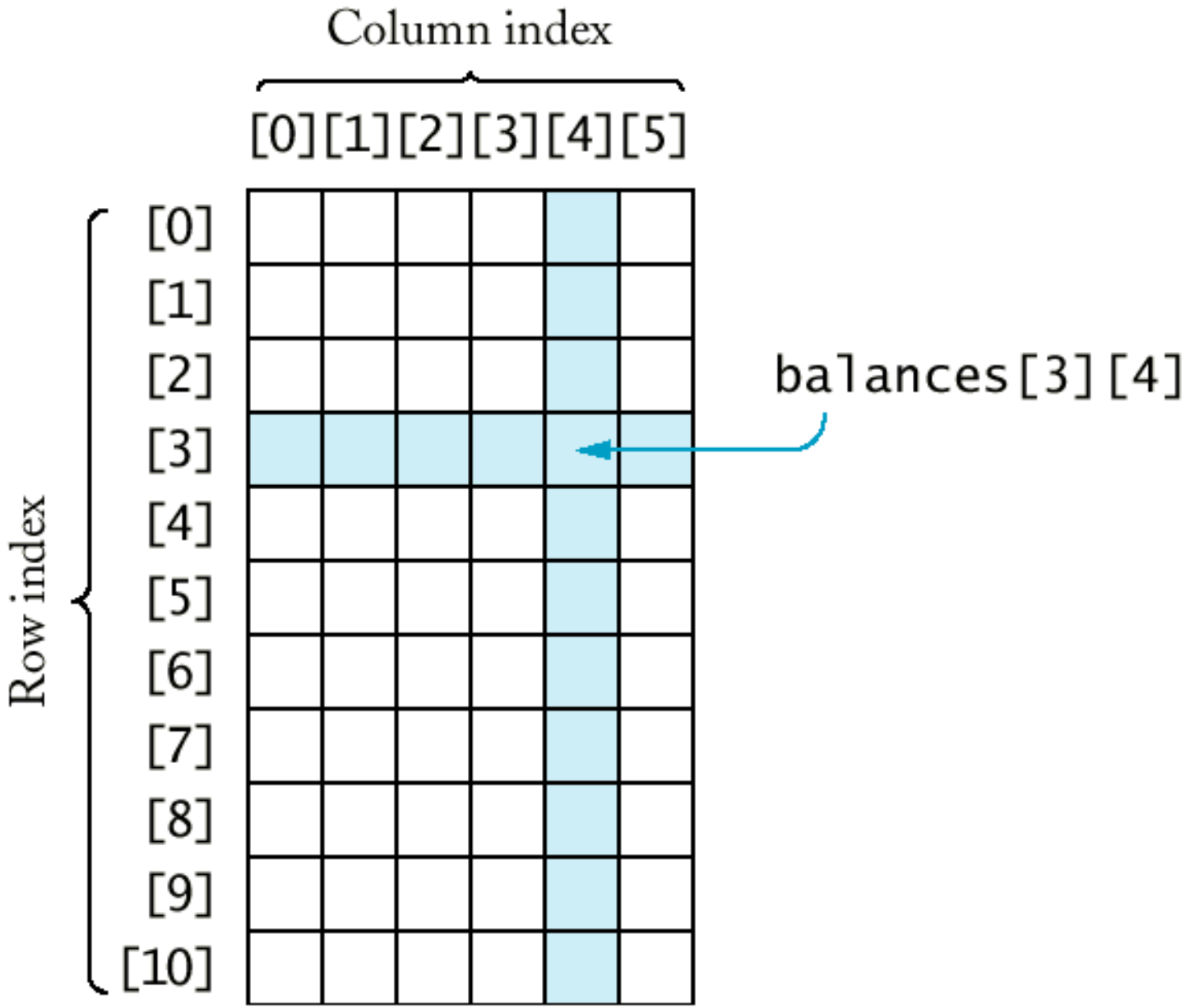
```
int numRows = 3;

int numCols = 3;

String[][] table =
           new String[numRows][numCols];

table[2][0] = “x”;
```




Column index
[0][1][2][3][4][5]
Row index
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]
balances[3][4]

2-D Array Question

```
public void alter(int c) {
    for (int i = 0; i<mat.length; i++)
        for (int j = c+1; j<mat[0].length; j++)
            mat[i][j-1] = mat[i][j];
}
```

mat is a 3x4 matrix with values:

```
1 3 5 7
2 4 6 8
3 5 7 9
```

alter(1) will change mat to what?

```
1 5 7 7
2 6 8 8
3 7 9 9
```

K-Graph

- Upper Triangle and Lower Triangle

SECTION 5

Unit 4
Object-Oriented Programming

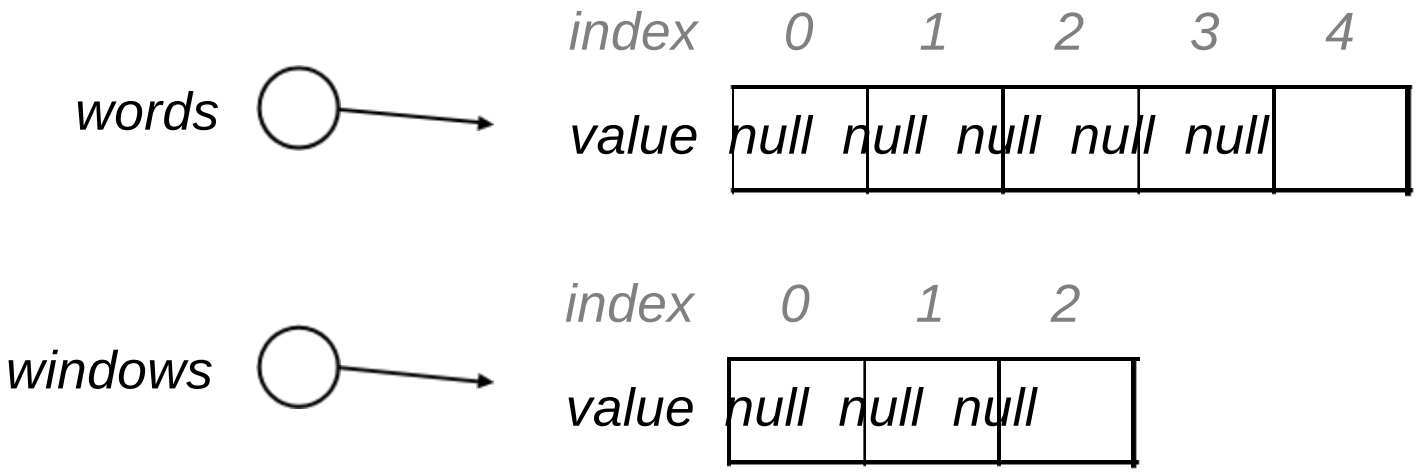
Objects, Classes, and Inheritance

- You may have to write your own class. You'll definitely need to interpret at least one class that's given. Very common, esp. on FRQ.
  - Methods
  - Subclasses
  - Abstract classes
  - Interfaces

Null

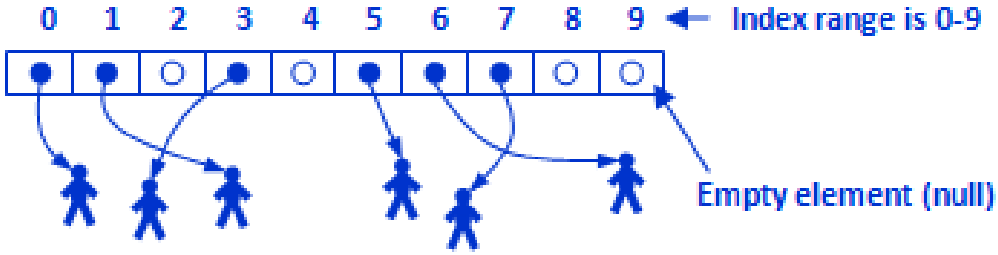
- **null :** A reference that does not refer to any object.
  - Fields of an object that refer to objects are initialized to null.
  - The elements of an array of objects are initialized to null.

```
String[] words = new String[5];
DrawingPanel[] windows = new DrawingPanel[3];
```



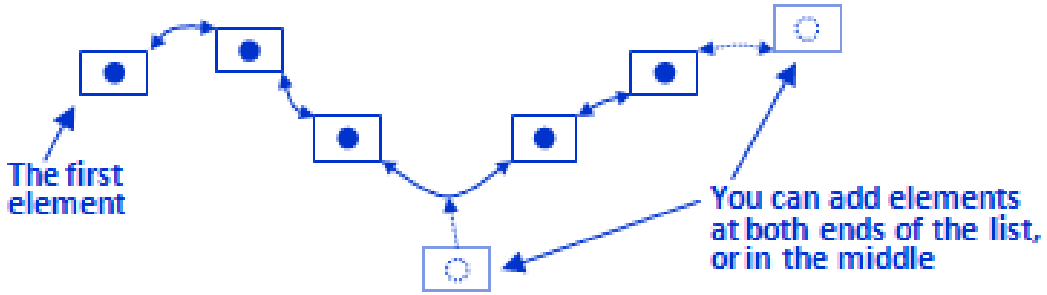
Array of 5 integers

Array of 10 agents



ArrayList (collection) of strings, currently contains 6 elements

LinkedList (collection)



Method Headers

With the exception of constructors, all method headers should have these 4 things:

- **Access modifier**: public, private
- **Return type**: void, int, double, boolean, SomeType, int[], double[], Pokemon[], etc.
- **Method name**: e.g. withdraw
- **Parameter list**: e.g. String pass, double amt
- (Some methods may also be static)

public void withdraw(String pass, double amt)

Types of Methods

Constructors → create an object of the class

- public BankAccount()

Accessor → gets data but doesn't change data

- public double getBalance()

Mutator → changes instance variable(s)

- public void deposit(String pswd, double amt)

Static methods → class methods, deals with class variables

- public static int getEmployeeCount()

Static methods in Driver class

- **Methods in the driver class** (the class that contains your main() method) are **usually all static** because there are not instances of that class.

public static void main(String[] args)

Method Overloading

- **Two or more methods with the same name but different parameter lists**
  - public int product(int n) {return n*n;}
  - public double product(double x) {return x*x;}
  - public double product(int x, int y){return x*y;}

- (return type is irrelevant for determining overloading)

Inheritance

- **Inheritance** is where a **subclass** is created from an existing **superclass**.
- The subclass copies or inherits variables and methods from the superclass
- Subclasses usually contain more than their superclass
- Subclasses can be superclasses for other subclasses

Class hierarchy - Which is true?

A. Superclass should contain the data and functionality that are common to all subclasses that inherit from the superclass

B. Superclass should be the largest, most complex class from which all other subclasses are derived

C. Superclass should contain the data and functionality that are only required for the most complex class

D. Superclass should have public data in order to provide access for the entire class hierarchy

E. Superclass should contain the most specific details of the class hierarchy

Implementing Subclasses

Subclasses copy everything except what?

```
public class Superclass {
    //superclass variables and methods
}

public class Subclass extends Superclass {
    //copies everything from Superclass
    //EXCEPT constructors
}
```

Inheriting Instance Methods/Variables

- Subclasses **cannot directly access private variables** if they are inherited from a superclass
- Subclasses must use the public accessor and mutator methods
- (Subclasses can directly access if variables are protected but protected is not in the AP Java subset.)

Method Overriding and super

- If a method has the same name and parameter list in both the superclass and subclass, the **subclass method overrides the superclass method**
- To invoke the method from the superclass, use the keyword **super**
- E.g. if the superclass has a computeGrade() method, use super.computeGrade()
- If you are invoking the constructor use super() or super(*parameters*)

Rules for Subclasses

- Can add new private instance variables
- Can add new public, private, or static methods
- Can override inherited methods
- May not redefine a public method as private
- May not override static methods of superclass
- Should define its own constructors
- Cannot directly access the private members of its superclass, must use accessors or mutators

Declaring Subclass Objects

- Superclass variables **can reference both superclass objects and subclass objects**
- Which of these is not valid:
  - Student c = new Student();
  - Student g = new GradStudent();
  - Student u = new UnderGrad();
  - UnderGrad x = new Student();
  - UnderGrad y = new UnderGrad();

Polymorphism

Method overridden in at least one subclass is polymorphic

What are method calls are determined by?

- the type of the actual object
- the type of object reference

Selection of correct method occurs **during the run** of the program (dynamic binding)

Type Compatibility

Only polymorphic if method is overridden

E.g. if GradStudent has a getID method but Student does not, this will lead to a compile-time error:

```
Student c = new GradStudent();
int x = c.getID();          //compile-error
```

You can cast it as a subclass object to fix the error:

```
int x = ((GradStudent) c).getID();
```

Abstract Class

- Superclass that represents an abstract concept
- Should never be instantiated
- May contain abstract methods
  - When no good default code for superclass
- Every subclass will override abstract methods
- If class contains abstract methods, it must be declared an abstract class

Notes about abstract classes

- Can have both abstract and non-abstract methods
- Abstract classes/methods are declared with keyword abstract
- Possible to have abstract class without abstract methods
- Abstract classes may or may not have constructors
- Cannot create abstract object instances
- Polymorphism still works

Interfaces

- Collection of related methods whose headers are provided without implementations
- Classes that implement interfaces can define any number of methods
- Contracts to implement all of them; if cannot implement all, must be declared an abstract class
- Interface keyword for interfaces; implements keyword for class that implement them
- Class can extend a superclass and implement an interface at the same time: public class Bee extends Insect implements FlyObject

Interface vs. Abstract Class

- Use abstract class for object that is application-specific, but incomplete without subclasses
- Consider interface when methods are suitable for your program but also equally applicable in a variety of programs
- Interface cannot provide implementations for any of its methods; abstract class can
- Interface cannot have instance variables; abstract class can
- Both can declare constants
- Both cannot create an instance of itself

Miscellaneous Note #2

List → basically the same as ArrayList

- List is an interface
- ArrayList implements List

SECTION 6

Algorithms

Sorting and Searching

Know these algorithms; at least one or two questions on AP exam

- Selection Sort
- Insertion Sort
- Merge Sort
- Binary Search

Binary Search

- Check middle element
  - Is this what we're looking for? If so, we're done.
  - Does what we're looking for come before or after?
- Throw away half we don't need
- Repeat with half we do need
- What does this look like in Java?

Binary Search Question

Which of the following is **not true** of a binary search of an array?

A. The method involves looking at each item in the array, starting at the beginning, until either the value being searched for is found or it can be determined that the value is not in the array.

B. In order to use the binary search, the array must be sorted first.

C. The method is referred to as “divide and conquer.”

D. An array of 15 elements requires at most 4 comparisons.

E. Using a binary search is usually faster than a sequential search.

How many executions?

- Check how many halves you threw away + 1
- Or check how many times you checked a middle element

Binary Search Question

A binary search is to be performed on an array with 600 elements. In the worst case, which of the following best approximates the number of iterations of the algorithm?

A. 6

B. 10

C. 100

D. 300

E. 600

Binary Search Question

Consider a binary search algorithm to search an ordered list of numbers. Which of the following choices is closest to the maximum number of times that such an algorithm will execute its main comparison loop when searching a list of 1 million numbers?

A. 6

B. 20

C. 100

D. 120

E. 1000

Selection Sort Algorithm (ascending)

"Search and swap" algorithm:

1. Find smallest element (of remaining elements).
2. Swap smallest element with current element (starting at index 0).
3. Finished if at the end of the array. Otherwise, repeat 1 and 2 for the next index.

Selection Sort Example(ascending)

70 75 89 61 37
- Smallest is 37
- Swap with index 0

37 75 89 61 70
- Smallest is 61
- Swap with index 1

37 61 89 75 70
- Smallest is 70
- Swap with index 2

37 61 70 75 89
- Smallest is 75
- Swap with index 3
  - Swap with itself

37 61 70 75 89
- Don’t need to do last element because there’s only one left

37 61 70 75 89

Selection Sort Question

In an array of Integer contains the following elements, what would the array look like after the third pass of selectionSort, sorting from high to low?

89 42 -3 13 109 70 2

A. 109 89 70 13 42 -3 2

B. 109 89 70 42 13 2 -3

C. 109 89 70 -3 2 13 42

D. 89 42 13 -3 109 70 2

E. 109 89 42 -3 13 70 2

Selection Sort Notes

- For an array of n elements, the array is sorted after n-1 passes
- After the kth pass, the first k elements are in their final sorted position
- Inefficient for large n

Insertion Sort Algorithm (ascending)

- Check element (store in temp variable)
- If larger than the previous element, leave it
- If smaller than the previous element, shift previous larger elements down until you reach a smaller element (or beginning of array). Insert element.

Insertion Sort Algorithm (ascending)

64 54 18 87 35

- 54 less than 64
- Shift down and insert 54

54 64 18 87 35

- 18 less than 64
- 18 less than 54
- Shift down and insert 18

18 54 64 87 35

- 87 greater than 64
- Go to next element

18 54 64 87 35

- 35 less than 87
- 35 less than 64
- 35 less than 54
- 35 greater than 18
- Shift down and insert 35

18 35 54 64 87

Insertion Sort Question

When sorted biggest to smallest with insertionSort, which list will need the greatest number of changes in position?

A. 5,1,2,3,4,7,6,9

B. 9,5,1,4,3,2,1,0

C. 9,4,6,2,1,5,1,3

D. 9,6,9,5,6,7,2,0

E. 3,2,1,0,9,6,5,4

Insertion Sort Question

- A worst case situation for insertion sort would be which of the following?
  1. A list in correct sorted order
  2. A list sorted in reverse order
  3. A list in random order

Insertion Sort Notes

- For an array of n elements, the array is sorted after n-1 passes
- After the kth pass, a[0], a[1],…, a[k] are sorted with respect to each other but not necessarily in their final sorted positions
- Worst case occurs if array is initially sorted in reverse order
- Best case occurs if array is already sorted in increasing order
- Inefficient for large n

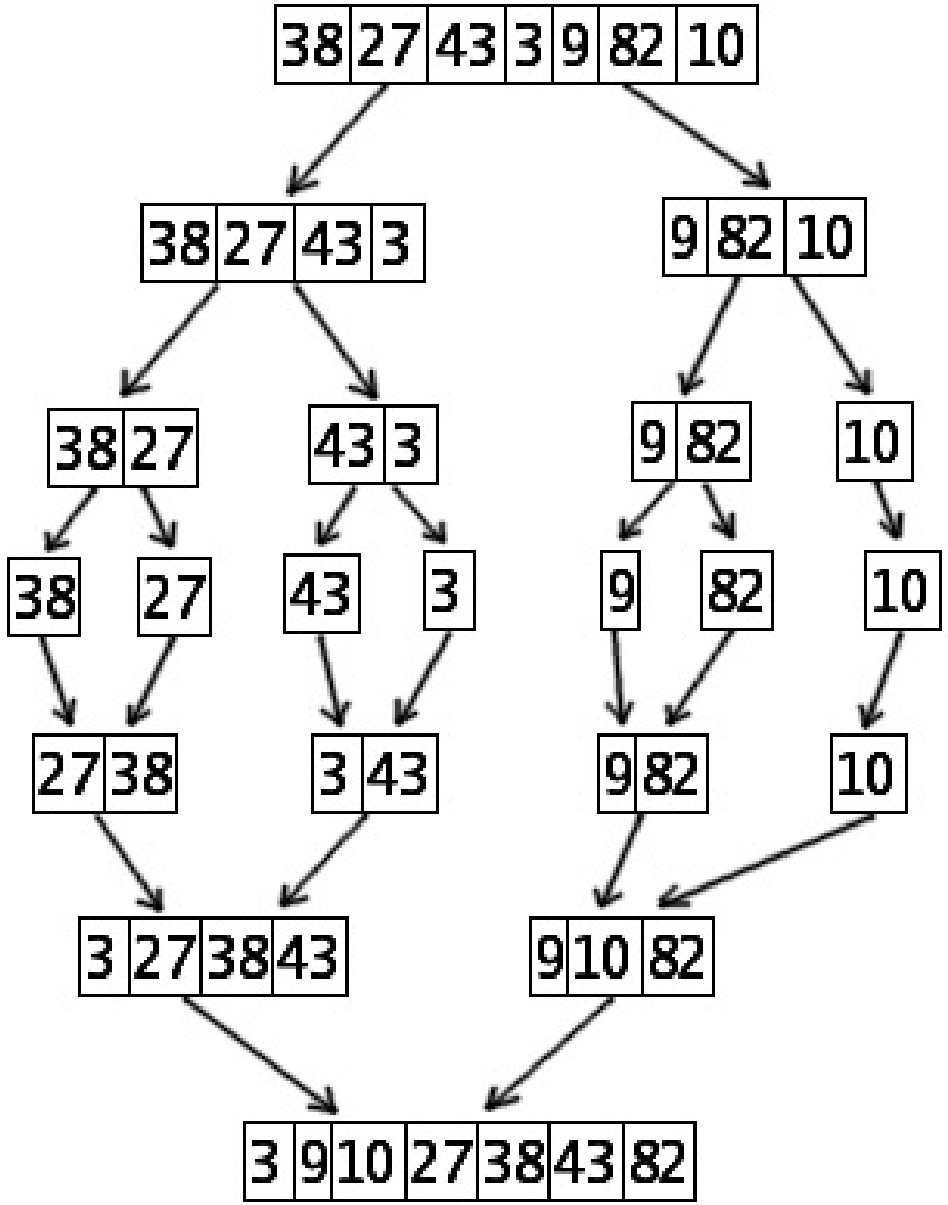
Merge Sort Algorithm

The idea behind merge sort is **divide and conquer**

1. Divide data into 2 equal parts
2. Recursively sort both halves
3. Merge the results

Merge Sort Example

1. Divide data into 2 equal parts
2. Recursively sort both halves
3. Merge the results



Merge Sort Example

8 45 87 34 28 45 2 32 25 78

8 45 87 34 28 | 45 2 32 25 78

8 45 87 | 34 28 | 45 2 32 | 25 78

8 45 | 87 | 34 | 28 | 45 2 | 32 | 25 | 78

8 | 45 | 87 | 34 | 28 | 45 | 2 | 32 | 25 | 78

8 45 | 87 | 28 34 | 2 45 | 32 | 25 78

8 45 87 | 28 34 | 2 32 45 | 25 78

8 28 34 45 87 | 2 25 32 45 78

- 2 8 25 28 32 34 45 45 78 87

Merge Sort Notes

- Main disadvantage of Merge sort is use of a temporary array→problem if space is a factor
- Merge sort is not affected by the initial ordering of the elements→best and worst case take the same amount of time

Merge Sort Question

- Which of the following is a valid reason why mergesort is a better sorting algorithm than insertion sort for sorting long lists?
  1. Mergesort requires less code than insertion sort
  2. Mergesort requires less storage space than insertion sort
  3. Mergesort runs faster than insertion sort

Recursive

- Binary search is recursive
- Uses indices to know where to search in the array
- Calculate an index midway between the two indices
- Determine which of the two subarrays to search
- Recursive call to search subarray

SECTION 7

Summary


May success be with you,
always...
Wishing you good luck!