Java Primitive Types

- Can be remember as Java native types
- These numeric types or character types are built in to help calculation
- Java primitive types are really sensitive to the value it get assigns to
    - A floating point number cannot be assigned to a int
    - A non-floating number cannot be assigned to a boolean
- Gramma matters
- Has default value

Java Statement

- A complete Java statement includes identifier, variable name, variable value, and a semicolon.
- A identifier defines the java type, Java primitive type can be a identifier.
- A variable name have to follow the java gramma standard
    - All alphabet (upper case and lower case) are allowed
    - Can connect alphabet with underscore _ or dollar sign $. All other symbols are illegal gramma
    - Number 0-9 are allowed, bet a initial letter is required. Just number alone is invalid
- To finish one Java Statement, the semicolon is a must.

Java Naming

- A variable name have to follow the java gramma standard
    - All alphabet (upper case and lower case) are allowed
    - Can connect alphabet with underscore _ or dollar sign $. All other symbols are illegal gramma
        - Number 0-9 are allowed, bet a initial letter is required. Just number alone is invalid
- Apply to
    - Variable
    - Method name
    - Class name

Numeric Primitive type

```
public class MyProgram {
        public static void main(String[] args) {

            int intValue = 10;
            short shortValue = 128;
            double doubleValue1 = 10d;
            double doubleValue2 = 10D;
            double doubleValue3 = 10.0;
            float floatValue1 = 10f;
            float floatValue2 = 10F;
            float floatValue3 = 10.0;

        }
}
```

boolean

- The boolean identifier is a Java primitive type

- A boolean variable can be assigned with true of false

Java Primitive Cast

- Cast the the operation to convert the target data type to the assigned data type
- <primitive type> var = (primitive type) targetVar
- e.g.
  - short sVar = 19;
  - int var = (int) sVar;

- Cast cross all the boundary. For safety use
  - Use cast follow by the assignment rule
- Cast floating points (safely) to non floating points will lose all the digit
- Cast non floating points (safely) to floating points, will add .0

```
double d = 888.0d;
long longValue = (long) d;
//longValue will be printed out with 888
```

```
int intValue = 223;
double d = (double) intValue;
//d will be printed out with 223.0
```

```
int intValue = 5;
double d =  (double) intValue  /  2;
```

2.5

```
int intValue = 5;
double d =  (double) (intValue  /  2);
```

2.0

boolean

```
public class MyProgram {
        public static void main(String[] args) {

boolean tVal = true;
boolean fVal = false;
// default false
boolean val;

        }
}
```

char

- Represent each single character of string

- Value from 0 to 255

ASCII Table

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(	72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29	)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[	123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	|
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

char

```
public class MyProgram {
        public static void main(String[] args) {

char c1= 45;
char c2 = ‘A’;
char c3 = ‘$’;

        }
}
```

Arithmetic Operators

- +: used for addition
- -: used for subtraction
- *: used for multiply
- /: used for division
- % (mod): used to get the remains of a division

Arithmetic Operators Question

```
public class MyProgram {
        public static void main(String[] args) {

                int a = 10;
                a += 10;

                a = a + 10;

        }
}
```

Special Arithmetic Operators

- ++
- - -

Special Arithmetic Operators

```
public class MyProgram {
        public static void main(String[] args) {

            int a = 10;
            a++;

            a = a + 1;

        }
}
```

Special Arithmetic Operators

```
public class MyProgram {
        public static void main(String[] args) {

            int a = 1;
            int b = 1;

            int c = ++b + a++;

        }
}
```

Relational Operators

- Execute the statement from left to right, relational operators give either true or false
- == : determine whether the left side is equals to the right side value
- !=: determine whether the left side is not equals to the right side value
- >: determine whether the left side is larger than the right side value
- <: determine whether the left side is smaller than right side value
- >=: determine whether the left side is larger or equals to right side value
- <=: determine whether the left side is smaller or equals to right side value

Logical Operators ||

Statement 1.	Statement 2.	Operator.	Value
true	true	||	true
true	false	||	true
false	true	||	true
false	false	||	false

One of the conditions need to be satisfied

Logical Operators &&

Statement 1.	Statement 2.	Operator.	Value
true	true	&&	true
true	false	&&	false
false	true	&&	false
false	false	&&	false

Both condition need to be satisfied

Logical Operators !

Statement	Operator.	Value
true	!	false
false	!	true

Opposite

Logical Operation Append Rules

Condition1 && Condition2 && Condition3 && ….

The more && statements get appended, the more strict the condition is

Condition1 || Condition2 || Condition3 || ….

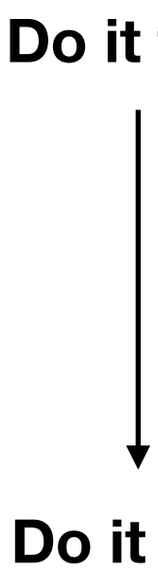
The more || statements get appended, the more flexible the condition is

DeMorgan's Law

$$!(a \;\&\&\; b) = !a \;||\; !b$$

$$!(a \;||\; b) = !a \;\&\&\; !b$$

Operators Computing Order



1. ! ++ —
2. * / %
3. + -
4. < > <= >=
5. == !=
6. &&
7. ||
8. = += -= *= /= %=

note: The horizontal order does not matter

Java numeric wrapper object

- Short
- Integer
- Long
- Float
- Double
- Boolean

Numeric Primitive type

```
public class MyProgram {
        public static void main(String[] args) {
                Integer intValue = 10;
                Short shortValue = 128;
                Double doubleValue1 = 10d;
                Double doubleValue2 = 10D;
                Double doubleValue3 = 10.0;
                Double doubleValue4 = new Double(10.0);
                Float floatValue1 = 10f;
                Float floatValue2 = 10F;
                Float floatValue3 = 10.0;
                Float doubleValue4 = new Float(10.0);
        }
}
```

Wrapper to primitive type

- This process is called box and unbox
- Wrapper class can be set to null

Object level access

```
public class MyProgram {
        public static void main(String[] args) {
                Integer intValue1 = 10;
                int toIntValue = intValue.intValue();
                short toShortValue = intValue.shortValue();
                double toDoubleValue = intValue.doubleValue();
                long toLongValue = intValue.longValue();
                String toStringValue = intValue.toString();
        }
}
```

Wrapper object level access

- All wrapper class provides function to covert to other primitive type value

- When your conversion is invalid, e.g. max integer to short, it will be over flow. So it still follows the primitive type casting rules

Class level access

```
public class MyProgram {
        public static void main(String[] args) {

            int maxValue = Integer.MAX_VALUE;
            int minValue = Integer.MIN_VALUE
            int convertValue = Integer.valueOf(“123”);

        }
}
```

Wrapper class level access

- Wrapper class provides lots of useful class level access utility function itself

- Instead of object level access, class level functions provides good error handling.

- When a invalid string is get converted, error will be thrown

Maths numerical operations packages

- A complete utility class, only provides class level access
- Provides tons of useful Maths operations
- We called these class pure utility classes

Exam related

- Math.random()
- Math.max(int a, int b)
- Math.min(int a, int b)
- Math.abs(int a)
- Math.round(float a)
- Math.floor(float a)
- Math.pow(double a, double b)

Math random min to max

min + (int)(Math.random * (max - min + 1))

Based calculation

Base 2

- Base: 0, 1
- Add the next significant digit when adding 1 to a ‘1’
- Most basic calculation unit of computing science

Base 8

- Base: 0, 1, 2, 3, 4, 5, 6, 7
- Add the next significant digit when adding 1 to a ‘7’
- Used in old computer systems

Base 10

- Base: 0, 1, 2, 3, 4, 5, 6, 7, 8 , 9
- Add the next significant digit when adding 1 to a ‘9’
- Foundation of maths

Base 16

- Base: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Add the next significant digit when adding 1 to a ‘F’
- Foundation of modern 64 bit and 32 bit OS systems

Convert Every other base representation to base 10

Value * base ^ (highest significantIndex)
+NextValue * base ^ (highest significantIndex - 1)
+....
+LastValue * base ^ (0)

1234_{10} to $????_{10}$

$$1 * (10 ^ 3) + 2 * (10 ^ 2) + 3 * (10 ^ 1) + 4 * (10 ^ 0) = 1234_{10}$$

1111_{2} to $????_{10}$

$$1 * 2 ^ 3 + 1 * 2 ^ 2 + 1 * 2 ^ 1 + 1 * 2 ^ 0 = 8_{10} + 4_{10} + 2_{10} + 1_{10} = 15_{10}$$

1234_{8} to $????_{10}$

$$1 * (8 ^ 3) + 2 * (8 ^ 2) + 3 * (8 ^ 1) + 4 * (8 ^ 0) = 668_{8}$$

1234_{16} to $????_{10}$

$$1 * (16 ^ 3) + 2 * (16 ^ 2) + 3 * (16 ^ 1) + 4 * (16 ^ 0) = 4660_{16}$$

$ABCD_{16}$ to $????_{10}$

$$A * (16 ^ 3) + B * (16 ^ 2) + C * (16 ^ 1) + D * (16 ^ 0) =$$
$$10_{10} * (16 ^ 3) + 11_{10} * (16 ^ 2) + 12_{10} * (16 ^ 1) + 13_{10} * (16 ^ 0) = 43981$$

Convert base 10 to every other base

- Step 1: Take the other base as divider
- Step 2: Use Decimal value mod divider
- Step 3: Write down mod value
- If the remaining decimal value is still larger than divider
  - Repeat step 2
  - otherwise write down last mod value

2	35	1
2	17	1
2	8	0
2	4	0
2	2	0
2	1	1

100011_2

8	35	3
8	4	4

43_8

16	35	3
16	2	2

23_{16}

Convert base 8 to binary

- The highest single digit of base 8 is 7
- 7 can be represent with just 3 bits: 111
- To convert base 8 to binary, calculate each digit to a 3 bits binary and combine them.
- e.g. $45_{8,}$ 4 convert to 100_2 , 5 to 101_2 result is 100101_2

Convert base 16 to binary

- The highest single digit of base 8 is F
- F maps to 15 in base 10,
- F can be represent with just 4 bits: 1111
- To convert base 16 to binary, calculate each digit to a 4 bits binary and combine them.
- e.g. $AB_{8,}$ A convert to 1010_2 , B to 1011_2 result is 10101011_2

Convert base 16 to base 8

- Convert base 16 to binary, calculate each digit to a 4 bits binary and combine them.

- Regroup the binary to a new group of 3 bits, make up the missing digits with 0

- e.g. AB_8, A convert to 1010_2, B to 1011_2 result is 10101011_2, Regroup $010/101/011_2$ result is 253_8

Convert base 8 to base 16

- Convert base 8 to binary, calculate each digit to a 3 bits binary and combine them.

- Regroup the binary to a new group of 4 bits, make up the missing digits with 0

- e.g. 253_8, result is 010101011_2, Regroup $0/1010/1011_2$ result is AB_8

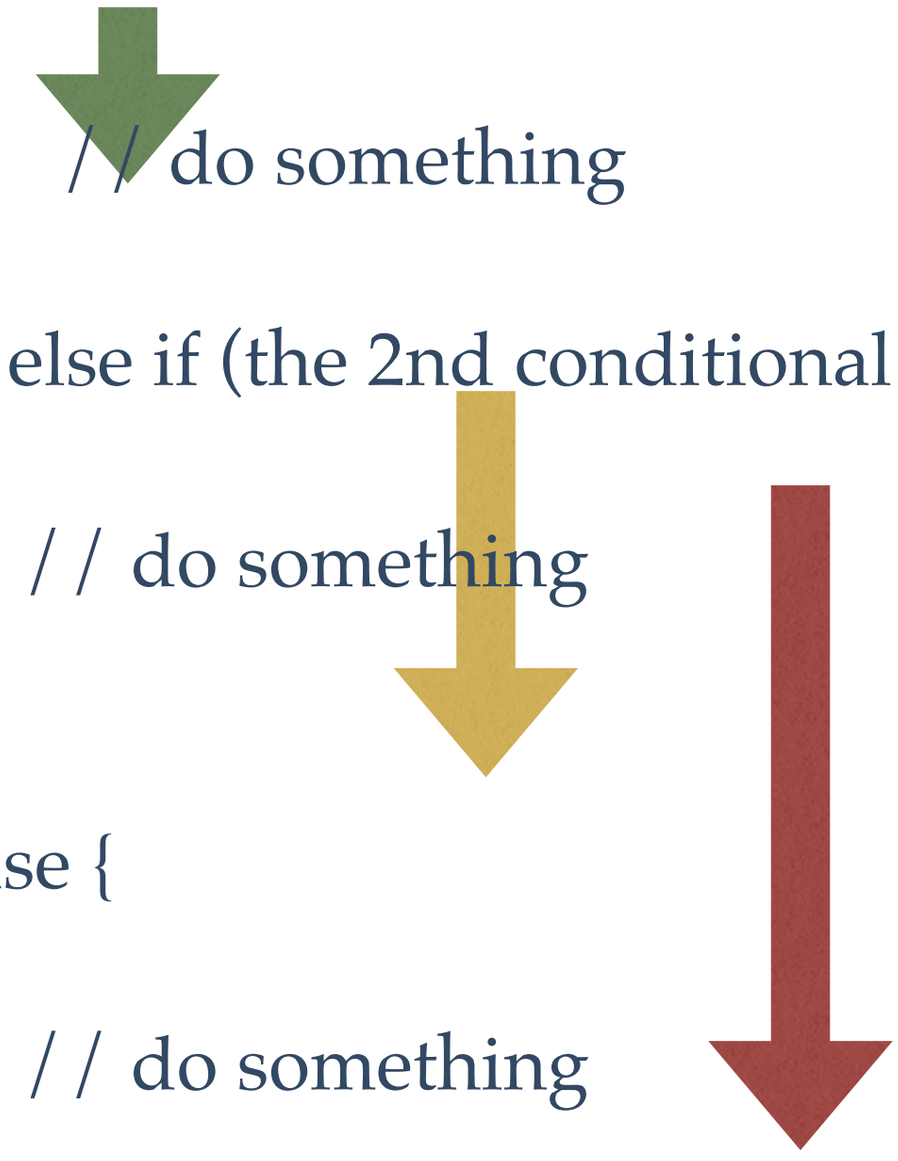
Java Primitive Assignment

- You can always assign value from a lower storage cost to a higher storage cost variable
- Floating points always larger than non floating points
  - You can assign any non floating points value to any floating points variable
  - No other way around

Java if else statement

- Control the where the next code execution goes to
- By given one or more logical statement to establish statement
  - if
  - if + else
  - if + else if + … + else
- Nest if else statement
  - The inner statement can be executed only if outer statement is passed
  - Nested statement can be understand as logical condition dependency

```
if ( the 1st conditional statement) {

    // do something

}  else if (the 2nd conditional statement) {

    //  do something

}
else {

    //  do something

}
```



Java Loop

For loop

One complete java statement

One conditional statement

One complete java statement
Or a java operation
Run after the loop operation


```
for ( [initial control variable declare]; [looping condition check] ; [condition change]) {

    // do something
}
```

While loop

One conditional statement

```
while ( [changeable condition]) {
        // keep doing something
}
```

Do while loop

```
do {

        // keep doing something

} while ( [changeable condition]);
```

One conditional statement

You can take the initiative and break the loop,
but you need to know what you are doing,
break should be used in side a if condition to be break properly

```
int myNumber = 1024;

while (myNumber > 0) {
    myNumber--;
    if (myNumber == 512) {
        break;
    }
}
System.out.println(myNumber);
```

Prints: 512

Nested for loop example

```
for (int i = 1; i <= 10 ; i++) {
    / / code here will be executed for 10 times (i times)
    for (int j = 1; j <= 5 ; j++)  {
    //code here will be executed for 50 times (i*j times)
    / / and yes, there are no restrict on how may nested for loops here
    }
}
```

For loop tips

- Do not modify the control variable (e.g. i or j) inside the loop body, it is very easy to mess up the loop logic

- Double check the control variable before you start the loop body writing, it is very easy to make a infinite loop

- Write down detailed steps if you are confused with the looping logic.

Java Variable Scope

Java Variable Scope

- Defines the variable accessibility level

- The deeper the variable get created, the less accessible it gets

```
public class MyProgram {
        public static void main(String[] args) {

            int myVariable = 6;

            if (myVariable >= 3) {
                // you can access myVariable here
                int innerVariable = myVariable;
                // you can only access innerVariable here inside if block
            }

             // you CANNOT access innerVariable here!!! EVER!!!!

            for (int i = 1; i <= 10 ; i++) {
                  // you can access i here
                  // you can access myVariable here
                  for (int j = 1; j <= 5 ; j++)  {
                      // you can access i here
                      // you can access j here
                      // you can access myVariable here
                  }
                  // you CANNOT access j here!!!!!
                  // you can access i here
            }

            // you CANNOT access  i  and j  here!!!!!
            // you can access myVariable here

        }  // nothing get access here
}
```

Java String

What is String

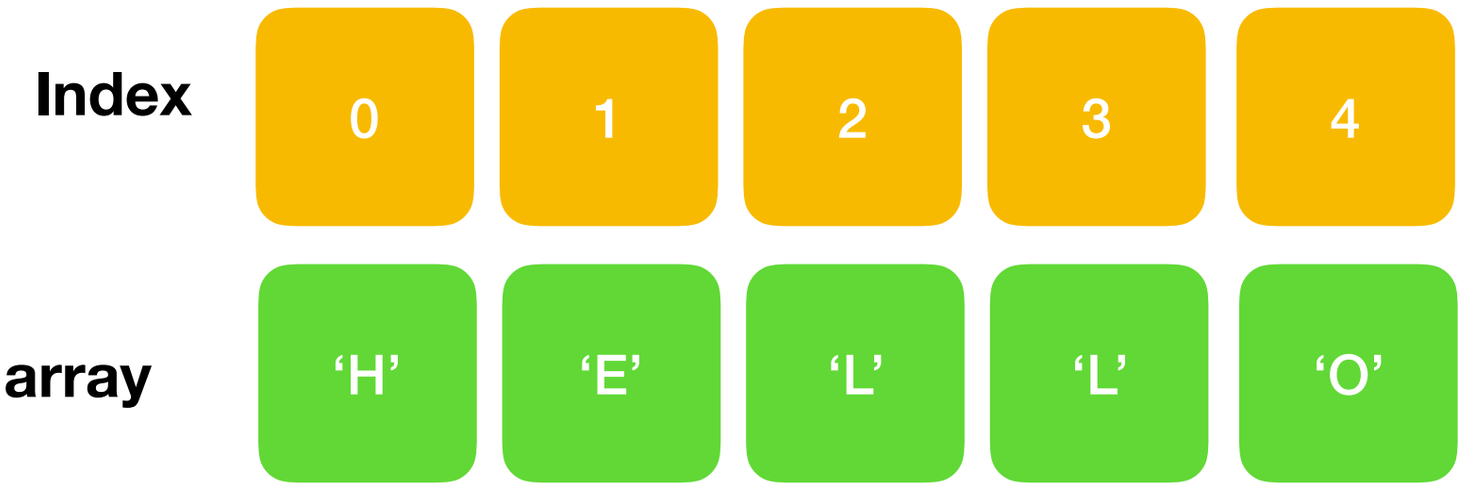
- String represent text form of data
- String is an object
- String is an array of char variables

Initial a String

```
public class MyProgram {
        public static void main(String[] args) {

            // Initial string with directly assign variable
            String value1 = “hello”;
           // Initial string with String class constructor
            String value2 = new String(“Hello”)

        }
}
```

String is a list of char

char: 16 bit - 2 bytes



String value “HELLO”

ASCII Table

Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char	Dec	Bin	Hex	Char
0	0000 0000	00	[NUL]	32	0010 0000	20	space	64	0100 0000	40	@	96	0110 0000	60	`
1	0000 0001	01	[SOH]	33	0010 0001	21	!	65	0100 0001	41	A	97	0110 0001	61	a
2	0000 0010	02	[STX]	34	0010 0010	22	"	66	0100 0010	42	B	98	0110 0010	62	b
3	0000 0011	03	[ETX]	35	0010 0011	23	#	67	0100 0011	43	C	99	0110 0011	63	c
4	0000 0100	04	[EOT]	36	0010 0100	24	$	68	0100 0100	44	D	100	0110 0100	64	d
5	0000 0101	05	[ENQ]	37	0010 0101	25	%	69	0100 0101	45	E	101	0110 0101	65	e
6	0000 0110	06	[ACK]	38	0010 0110	26	&	70	0100 0110	46	F	102	0110 0110	66	f
7	0000 0111	07	[BEL]	39	0010 0111	27	'	71	0100 0111	47	G	103	0110 0111	67	g
8	0000 1000	08	[BS]	40	0010 1000	28	(	72	0100 1000	48	H	104	0110 1000	68	h
9	0000 1001	09	[TAB]	41	0010 1001	29	)	73	0100 1001	49	I	105	0110 1001	69	i
10	0000 1010	0A	[LF]	42	0010 1010	2A	*	74	0100 1010	4A	J	106	0110 1010	6A	j
11	0000 1011	0B	[VT]	43	0010 1011	2B	+	75	0100 1011	4B	K	107	0110 1011	6B	k
12	0000 1100	0C	[FF]	44	0010 1100	2C	,	76	0100 1100	4C	L	108	0110 1100	6C	l
13	0000 1101	0D	[CR]	45	0010 1101	2D	-	77	0100 1101	4D	M	109	0110 1101	6D	m
14	0000 1110	0E	[SO]	46	0010 1110	2E	.	78	0100 1110	4E	N	110	0110 1110	6E	n
15	0000 1111	0F	[SI]	47	0010 1111	2F	/	79	0100 1111	4F	O	111	0110 1111	6F	o
16	0001 0000	10	[DLE]	48	0011 0000	30	0	80	0101 0000	50	P	112	0111 0000	70	p
17	0001 0001	11	[DC1]	49	0011 0001	31	1	81	0101 0001	51	Q	113	0111 0001	71	q
18	0001 0010	12	[DC2]	50	0011 0010	32	2	82	0101 0010	52	R	114	0111 0010	72	r
19	0001 0011	13	[DC3]	51	0011 0011	33	3	83	0101 0011	53	S	115	0111 0011	73	s
20	0001 0100	14	[DC4]	52	0011 0100	34	4	84	0101 0100	54	T	116	0111 0100	74	t
21	0001 0101	15	[NAK]	53	0011 0101	35	5	85	0101 0101	55	U	117	0111 0101	75	u
22	0001 0110	16	[SYN]	54	0011 0110	36	6	86	0101 0110	56	V	118	0111 0110	76	v
23	0001 0111	17	[ETB]	55	0011 0111	37	7	87	0101 0111	57	W	119	0111 0111	77	w
24	0001 1000	18	[CAN]	56	0011 1000	38	8	88	0101 1000	58	X	120	0111 1000	78	x
25	0001 1001	19	[EM]	57	0011 1001	39	9	89	0101 1001	59	Y	121	0111 1001	79	y
26	0001 1010	1A	[SUB]	58	0011 1010	3A	:	90	0101 1010	5A	Z	122	0111 1010	7A	z
27	0001 1011	1B	[ESC]	59	0011 1011	3B	;	91	0101 1011	5B	[	123	0111 1011	7B	{
28	0001 1100	1C	[FS]	60	0011 1100	3C	<	92	0101 1100	5C	\	124	0111 1100	7C	|
29	0001 1101	1D	[GS]	61	0011 1101	3D	=	93	0101 1101	5D	]	125	0111 1101	7D	}
30	0001 1110	1E	[RS]	62	0011 1110	3E	>	94	0101 1110	5E	^	126	0111 1110	7E	~
31	0001 1111	1F	[US]	63	0011 1111	3F	?	95	0101 1111	5F	_	127	0111 1111	7F	[DEL]

String length()

```
public class MyProgram {
        public static void main(String[] args) {

            // Initial string with directly assign variable
            String value = “hello”;
            // size is 5
            int size = value.length();

        }
}
```

String toCharArray()

```
public class MyProgram {
        public static void main(String[] args) {

            // Initial string with directly assign variable
            String value = “hello”;
            // value list is h, e, l, l, o
            char[] valuelist = value.toCharArray();

        }
}
```

String charAt(int index)

```
public class MyProgram {
        public static void main(String[] args) {

            // Initial string with directly assign variable
            String value = “hello”;
            // valueChar is  ‘o’
            char valueChar = value.charAt(4);

        }
}
```

String upper/lower case

```
public class MyProgram {
        public static void main(String[] args) {

            // Initial string with directly assign variable
            String value = “Hello”;
            // upper is  HELLO
            String upper = value.toUpperCase();
            // lower is hello
            String lower = value.toLower();

        }
}
```

String subString

```
public class MyProgram {
        public static void main(String[] args) {

            // Initial string with directly assign variable
            String value = “Hello”;
            // upper is llo
            String upper = value.subString(2);
            // lower is Hel
            String lower = value.toLower(0, 2);

        }
}
```

System.out.print rule1

Calculate

Append

number + number + string + number + number

System.out.print rule1 Escape

\ Special charactor

“\”time\””

- \” print “
- \’ print ‘
- \t print tab
- \n print next line

Java Method

Java Method

- Can see as a function
  - You give input, the function gives output
  - stateless: the same input always gives same output
- Java Method is re-useable
- Can be assign to a Java variable
- If the method returns a value, it can be directly called in System.out.print
- Method can call other method

Format

```
[Modifier] [static/non static] [return type] [method name] ([parameter1], [parameter2] …) {
        // Method body
        return [return value]
}
```

Java Method Rule

- Naming convention follows Java variable, you can connect words with _ and $, you can include numbers in the name, but the method name has to start with a letter
- Method has to have a return type.
- The final return type needs to match the define return type
- Method body needs to be wrapped by {}
- Method itself will execute, method needs to be called or in official term “invoked”

Java Method Parameter Rule

- Technically, you can have unlimited method parameter
- Parameter declare follows normal Java variable rules
- Parameter can directly be used in the method body
- Method can have zero parameter

A void method

- If there is nothing to return in the method. Define void
- Java main method is a void method

Void method

```
public class MyProgram {
        public static void main(String[] args) {
            functionVoid(1, 3);
        }
        public static void functionVoid(int a, int b) {
            System.out.println(a + b);
        }
}
```

method with return

```
public class MyProgram {
        public static void main(String[] args) {
            int c = sub(addition(1,2), sub(3, 4));
        }
        public static int addition(int a, int b) {
            return a + b;
        }
        public static int sub(int a, int b) {
            return a - b;
        }
}
```

Object Oriented Programming

- Java object is mapping the the O in the OOP
- Java is pretty much build with classes and objects
- Object and classes gives a way to describe the programming objects
- It contains instance variables, method and programming logics

public ? private

- Java key word to describe visibility
- public
  - Instance or method has global access outside class
- Private
  - Instance or method has limit access only inside class
- No define
  - By default, instance or method is private

A Java Constructor

- A special type of Java method
    - Every class most, and at least have one
    - Follow all java method rule
    - Can have parameter
    - Can have implication body
- Callers call this to initial the java object

A little more on Object Oriented Programming

- With Object it pretty much open up all possibility of Java programming

- Object could contains other object instance for more complicate logic

- You can use objects just like java primitive type

  - Compare them (will cover later)

  - Pass them in as a parameter

Object Oriented Programming Best Practice

- As object is a way to describe the programming target
  - Think through every detail of your target
  - Think about the target behaviour
- Before: more algorithm thinking
- Now: more design thinking
- Instance variables: describe what the class have
- Methods: describe what the class can do

Java static

Can be used to define a variable
Can be used to define a method
When marked as static, the variable or method has express access or aka class level access

Class level access
Vs
Object level access

Class level access:

Classname.variableName

Classname.methodName

Object level access:

Need to create the object first

ClassName objectName = new ClassName();

objectName.instanceName

objectName.methodName

Cross Class Relationship

- Class could have other class as its instance attribute

- Method of a class can take other class type as a parameter

- Method in a class can return other class as a return type

Array Summary

Array

- Represent a list of Same data type
- Everything can be represent as array in Java
- Created with a initial value
- Is an “object” so it will allocate memory

Array Format

- You can have
  - int[], float[], double[]
  - boolean[], String[]
  - The object you created array
    - Student[], Score[]

Initial an array

- When array is initial created it is empty
- It just provides a container to store value
- For primitive type all value goes to the default when created
  - int, short, long, default value is 0
  - float, double default value is 0.0
  - boolean is false
  - The object you created is null value

Access an Array

- Access an element in array with its index

- Array start with index 0

Go through an array

- Looping with for or while
- You can start at any index
- DO NOT go across the index boundary
- The index boundary is array length - 1

Index	0	1	2	3	4	5	6	7
int array	7	4	5	3	11	9	3	1

Example Code

```
public class MyProgram {
        public static void main(String[] args) {

                // create an int array of size 10. You can store 10 integer here
                int[] intArray = new int[10];

        }
}
```

Example Code

```
public class MyProgram {
        public static void main(String[] args) {

            // create an int array of size 10. You can store 10 integer here
            int[] intArray = new int[2];
            intArray[0] = 1;
             intArray[1] = 2;

        }
}
```

Example Code

```
public class MyProgram {
        public static void main(String[] args) {

            // create an int array of size 10. You can store 10 integer here
            int[] intArray = new int[2];
            intArray[0] = 1;
            intArray[1] = 2;
            System.out.println(intArray.length);

        }
}
```

Print out 2

Since Array is an object Important attribute

- object.length

```
public class MyProgram {
        public static void main(String[] args) {

            // create an int array of size 10. You can store 10 integer here
            int[] intArray = new int[10];

            for (int i = 0; i < intArray.length; i ++) {
                intArray[i] = i  + 1;
            }

            for (int j = 0; j < intArray.length; j ++) {
                System.out.println(intArray[j] + " ");
            }

        }
}
```

Print out 1 2 3 4 5 6 7 8 9 10

Array, Object Memory Reference Summary

Primitive Type

- Primitive type does not go into Java heap memory
- When used by functions, only the value is passed in
- Primitive type value will not be influenced by method

```
public class MyProgram {
        public static void main(String[] args) {
                int a = 10;
                int b = function(a);
                System.out.print(a);
                System.out.print(b);
        }

        public int function(int a) {
                a = a + 10;
                return a;
        }
}
```

Object Type

- Object types are stored in Java heap memory

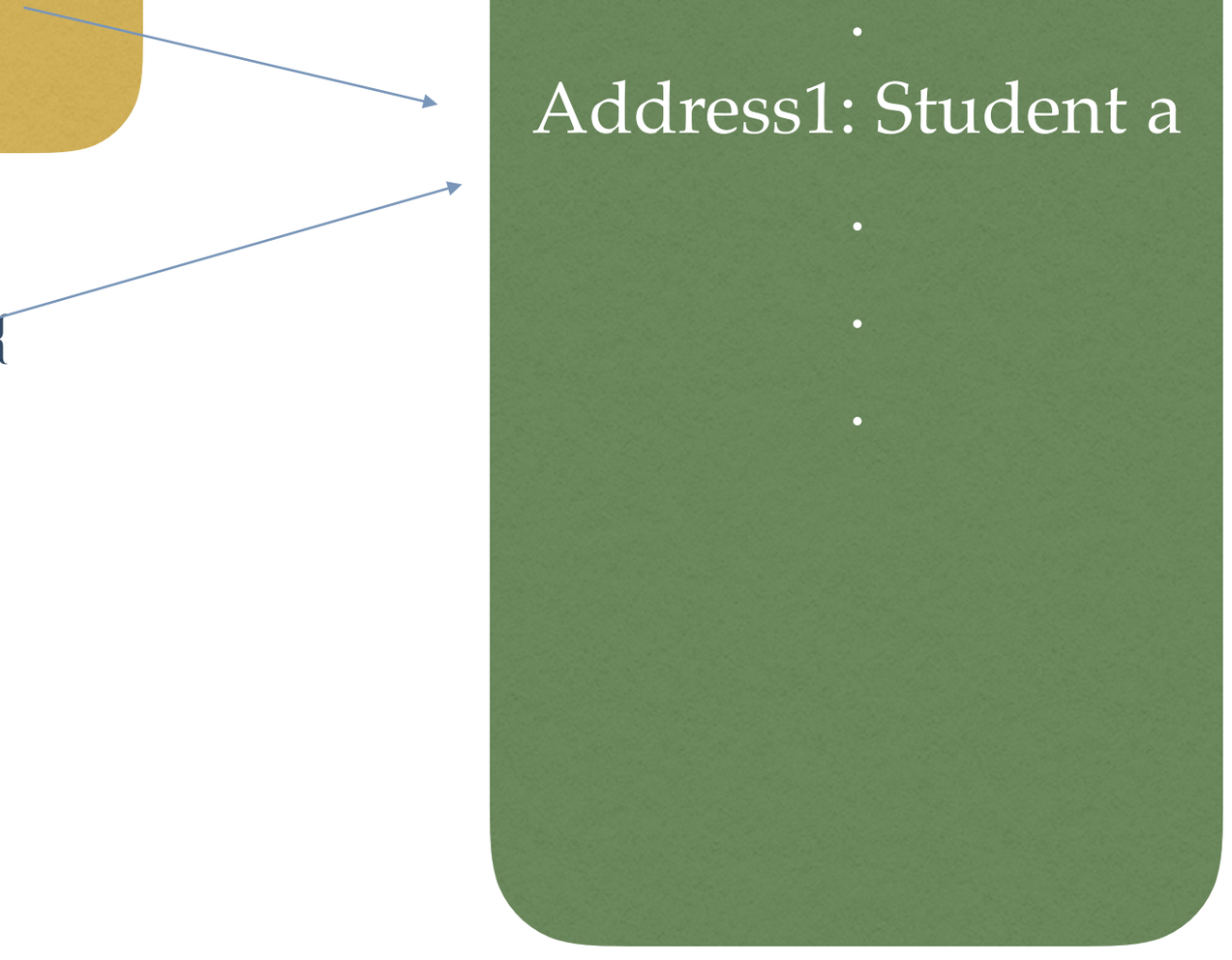
- When used by functions, the memory reference is passed in to the function

- Object type value will not be influenced by method, be careful

```
public class MyProgram {
        public static void main(String[] args) {
                int finalScore = 90;
                Student a = new Student(90);
                function(a);
                System.out.print(a.final);
        }

        public void function(Student a) {
                a.finalScore = 100;
        }
}
```

shallow copy

```
public class MyProgram {
        public static void main(String[] args) {
                int finalScore = 90;
                Student a = new Student(90);
                function(a);
                System.out.print(a.final);
        }

        public void function(Student a) {
                a.finalScore = 100;
        }
}
```



```
public class MyProgram {
        public static void main(String[] args) {
                int[] myarray = {1,2,3,4,5};
                function(myarray);
                System.out.print(myarray[0]);
        }

        public void function(int[] input) {
                input[0] = 9;
        }
}
```

Object Copy

- Since Object types are stored in Java heap memory
- Directly use "=" will just point to the same memory address, this is shallow copy
- To deep copy, make sure all object values are copied and created a new object using new key word.
- For array, each value should be copied to the new array spot

Java 2D array

Array

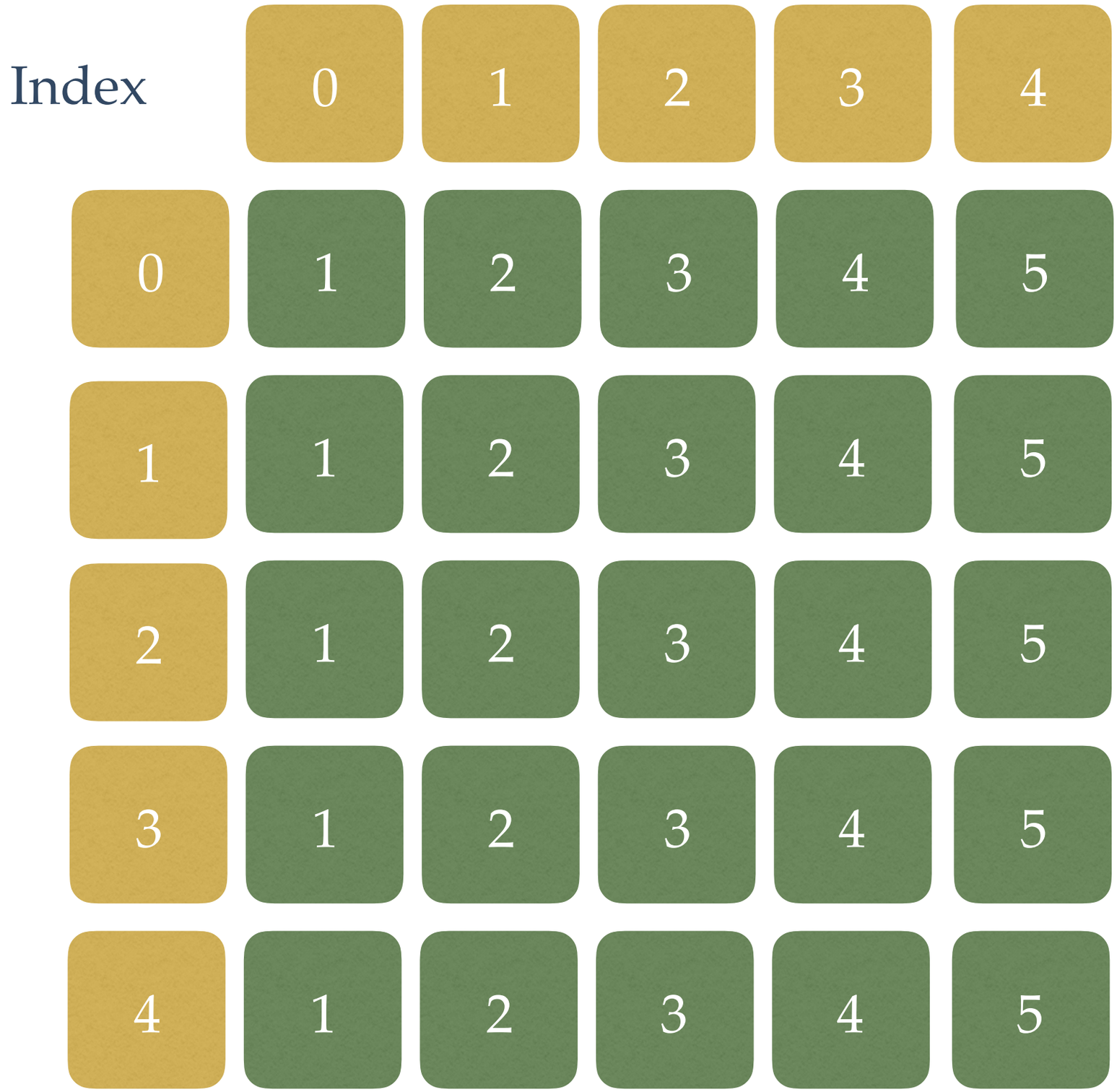
- Represent a list of Same data type
- Everything can be represent as array in Java
- Created with a initial value
- Is an “object” so it will allocate memory

An array of array - 2D array
Or Matrix

- Represent a grid of data
- Data spread in 2 dimension
- A grid of same type of data

Access an 2D array

- Access an element in array with its index
- Access an element in 2d array with its coordinate


Index
0
1
2
3
4
0
1
2
3
4
5
1
1
2
3
4
5
int matrix
2
1
2
3
4
5
3
1
2
3
4
5
4
1
2
3
4
5

Important Matrix attribute

- int[][] matrix
- matrix.length return number of rows in matrix
- matrix[index].length return number of columns in matrix

```
public class MyProgram {
        public static void main(String[] args) {

            // create an 4*5 matrix
            int[][] matrix = new int[4][5];

            // rows is now equals to 4
            int rows = matrix.length;

            // columns is now equals to 5
            int columns = matrix[0].length;

        }
}
```

Go through a matrix

- Looping with nested for loop
- Control index with meaningful variable name
- DO NOT go across the index boundary

```
public class MyProgram {
        public static void main(String[] args) {

            // create a 3*3 matrix
            int[] matrix = new int[3][3];


            for (int row = 0; row < matrix.length; row ++) {
                for (int column = 0; column < matrix[0].length; column++) {
                        System.out.println(matrix[row][column]);
                }
            }

        }
}
```

Print out 1 2 3 4 5 6 7 8 9 10

Java ArrayList

Basic Type Array Problems

- Need a for loop to do everything
- Hard to copy an array into another
- Hard to remove items
- Need to create with a initial size

Java ArrayList

- A Java built in data structure collection
- Also a container for a list of same typed object
- A Java generic template
- Not for primitive type
- Provided way more functions

ArrayList Format

- ArrayList<ObjectType> variableName = new ArrayList<ObjectType>();
- <> represent as this is a Java generic template collection
- ObjectType defines what object can be put into array list

Example ArrayList

```
public class MyProgram {
        public static void main(String[] args) {

            // create an int array list
            ArrayList<Integer> list = new ArrayList<Integer>();

        }
}
```

Access an ArrayList

- Access an element in arraylist is like access them in array
- Use .add(Object target) to add at tail of list
- Use.remove(Object target) to remove an object
- Use .get(int index) to access element
- Use .set(int index, Object target) to override an element
- When arraylist is created, this is also empty

Add item in ArrayList

```
public class MyProgram {
        public static void main(String[] args) {

            // create an int array list
            ArrayList<Integer> list = new ArrayList<Integer>();
            Integer input = 10;
            list.add(input);

        }
}
```

ArrayList size

```
public class MyProgram {
        public static void main(String[] args) {

            // create an int array list
            ArrayList<Integer> list = new ArrayList<Integer>();

            int currentSize = list.size();
            list.add(10);
            list.add(12);

            currentSize = list.size();

        }
}
```

Remove by index in ArrayList

```
public class MyProgram {
        public static void main(String[] args) {

            // create an int array list
            ArrayList<Integer> list = new ArrayList<Integer>();

            int currentSize = list.size();
            Integer input1 = 10;
            Integer input2 = 12;
            list.add(input1);
            list.add(input2);

           list.remove(0);

        }
}
```

Remove by item in ArrayList

```
public class MyProgram {
        public static void main(String[] args) {

            // create an int array list
            ArrayList<Integer> list = new ArrayList<Integer>();

            int currentSize = list.size();
            Integer input1 = 10;
            Integer input2 = 12;
            list.add(input1);
            list.add(input2);

           list.remove(input1);

        }
}
```

Go through ArrayList

```
public class MyProgram {
        public static void main(String[] args) {

                // create an int array list
                ArrayList<Integer> list = new ArrayList<Integer>();

                for ( Integer item : list) {
                        System.out.println(item);
                }

        }
}
```

Algorithm

- Find Max
- Find Min
- Go through array, 2D array, forward/backward
- Build array, 2D array, forward/backward