Study notes on programming and object-oriented programming

Chapter 1 INTRODUCTION TO COMPUTER

1. Common thing among programming language
   a) Puntuation - most programming language use puntation.
   b) Syntax - Allocate how program is written
   c) Operator - used to perform operation on two operand
   d) Keyword -

2. Example of program in Java

```
public class Main {
    public static void main(String[] args) {
        int hours = 2;
        double hourlyPay = 9;
        double pay = hours * hourlyPay;
    }
}
```

3. Variable store an item of data in memory

Example

```
int pay = 9;
```

4 Java file are save with the extention .java

5. Ways to design a program
   a) Clearly define what program is to do
   b) Vistualize the program running

6. Object oriented programming is the creation of data and procedure which works on the data.

CHAPTER 2 JAVA FUNDAMENTAL

1. Simple example of a Java program

```
public class Main {
    public static void main(String[] args){
        System.out.println("fun");
    }
}
```

name of method

There most only be one public class

This is used to print a statment and nextline

2. Java is case sentitive.

3. Statement are terminated with a semicolon.

4. Special character
   a) // - Mark comment beginning
   b) { } - enclose class and content of method
   c) ; - mark end of statement.

5. System.out.println(""); ← produce output and move to nextln
   System.out.print(""); ← Just produce output

   ↓
   The system class hold the object out that produce the output to the screen

6. Common escape sequence
   - it perform a specific tasks.

   Example

   System.out.print("Sarah\n");

   ↑ move to next line

7. Common escape sequence
   1) \' - Single quote

2) \" - double quote

3) \\ - backslash.

4) \n - nexline

8. Variable is a storage location of data.

Example

```
public class Main {
    public static void main(String[] args){
        int value;
        value = 2;
        System.out.println(value);
    }
}
```

This is a variable declaration → `int value;`

assignment statment → `value = 2;`

`value` → variable

9. String literal is the word surrounded by double quotation it is display how it appear.

Example

```
System.out.print("value");
```

`"value"` → string literal

output - value

10.) + operator for displaying multiple item

Example

```
System.out.println("iyanu" + "is a boy");
```

`"iyanu" + "is a boy"` → append string literal

```
System.out.println("iyanu" + value);
```

append a string literal to a variable

11) Variable name rules
- it can't start with digit
- it can't contain space
- it can contain alphabet, digit, underscore

Example

int 3day - illegal
int day week - illegal
int _day - legal

12) There are different data type which hold different value

Example

byte - -128 to +127
short - -32,768 to +32,767
int - -2,147,483 to +2,147,483,647
float - $\pm 3.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$
long - $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$

13) Data type

Example

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

```
        int value;
        double num1;

        value = 1280;
        num1 = 12.99;
    }
}
```

14 char Data type – it is used to hold character. it has a single quotation

Example

```
char letter;
letter = 'A';
```

15. Boolean data type hold to possible value true or false

Ex

```
boolean bool;
bool = true;
```

16. Arithemetic Operator
    - it is used to perform arithmetic calculation.

Exmple

+, -, *, /, %
(/ division, % remainder)

17. Example of Arithmetic operation

Example

```
int value1 = 12;
int value2 = 3;
int value3 = 7;
double value;

value = value1 + value2;
System.out.println(value);
value = value1 - value2;
System.out.println(value);
value = value1 / value2;
System.out.println(value);
value = value1 %value3;
System.out.println(value);
```

output:
15

output:
9

output:
4

output:
5

output
15
9
4
5

18) Precedence of arithmetic operator : which one come first

From the left to right

- (unary operator)
* / %
+ -

Highest to lowest

19) Group with Parethesis
- It force some operation to be performed first

Example

```
average = (a + b + c + d)/4.0;
```

perform together

20) Math class used to perform complex math operation which is in the Java API.

Example

```
result = Math.pow(2, 4);    2^4
result = Math.sqrt(9);      3
```

pow ↙

↳ square root

21) Cast operator manually let you convert a value

Example

```
int pies=10, people=4;
double piePerPerson;
piePerPerson = pie /(double) people;
```

output:

2.5

22) Creating a constant variable

```
final double INTREST_RATE=10;
```

↳ constant

23) String class is provided by Java API which hold string

Example

```
String name;
```

↳ in capital letter

24) Primative data type - hold actual data item
Class type variable - hold memory address of data item.

25) Class name is started with capital letter

Example

```
String name;
  ↓ capital
System.out.print( );
  ↓ capital
```

26) Creating string object

Example

```
String name = "Jame";
```

string object "Jame"

name ⟶ hold address of string object

27) Few String class method

- To call a method

referenceVariable.method(argument);

- charAt( ) - get specific index
- length( ) - length of string
- toLowerCase( ) - change string to lower case

Example

```
String name = "sarah";
int length;
length = name.length();
```

28) Scope of a variable is that it can't be accessed by code outside method or inside method but before variable declaration.

Example

```
System.out.println(value); ← Error
int value = 5;
```

29) Comment - are ignored by complier which is used to explain code

Example

```
// Program Comment1.java
Public class Comment1 {
    public static void main (String[] args){
        double rate; // This is Rate
                     ↑ it ignored by Complier
    }
}
```

30) Type of comment

- Single line Comment: // for a singl line
- Multiple line Comment: /* (start) ... */ (end)

31) Programming Style refer to ways programmer write code

- Indent all line in a set of braces

Example

```
public class Grade {
    public static void main(String[] args){
        int Grade=2;
    }
}
```

indented → `public static void main(String[] args){`

indented → `int Grade=2;`

32) Scanner class is used to read input from a source (System.in) and provide method to retrieve it as primative data type or strings.

Example

```
Scanner keyboard = new Scanner(System.in);
```

```
Scanner keyboard = new Scanner(System.in);
```

keyboard: the variable name reference the scanner object

new Scanner(System.in): This read input from the system.in

```
int number=keyboard.nextInt();
```

3.3) To use the scanner class we have to include the header file

- import java.util.Scanner
  (S: capital)

3.4) Some Scanner class method

- nextDouble(); ← read input as double
- nextInt(); ← read input as int
- nextLine(); ← read input as string
  - Allows include nextline() to use

.nextInt and .nextDouble

This take input and create a Line break and can pass through Line break.

Example

```
Scanner keyboard = new Scanner(System.in);

System.out.println("Enter number");
int number1 = keyboard.nextInt();

System.out.println("Enter number 2");
int number2 = keyboard.nextInt();
System.out.println(number1+" "+number2);
```

This create a line break in buffer

This is able to skip a line breaks in buffer

input:

Ente a number
1

Enter a number 2
2

output:

1 2

.nextLine

This take input and doesn't create a line break an d never skip a line break

Example

```
Scanner keyboard = new Scanner(System.in);
```

```
System.out.println("Enter number");
int number1 = Keyboard.nextInt();
Keyboard.nextLine();
System.out.println("Enter a name");
String name = Keyboard.nextLine();
System.out.println(number1 + " " name);
```

Create Line break

This help SKip line break

This neve skip line break

input :
1
Victor

output :
1 Victor

Example

```
Scanner Keyboard = new Scanner(System.in);
System.out.println("Enter a name");
String name = Keyboard.nextLine();
System.out.println("Enter number");
int number1 = Keyboard.nextInt();
System.out.println(number1 + " " name);
```

There is no need for what skip a line break because there is no line break in buffer

35) Alway close the scanner class
Keyboard.close();

36) To use the Message Dialog box or input Dialog box we have to import the JOptionPane class.

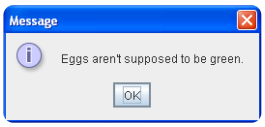
```
import javax.swing.JOptionPane;
```

37) Displaying a message Dialog box

Example

```
JOptionPane.showMessageDialog(null, "Egg or ep't suppose to be green");
```

This is what Displayed



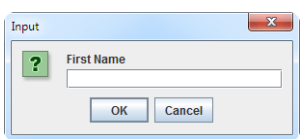
Cause dialog box to be display at center.

3) Input Dialog box

- It is used to get user input which is in form of string.

Example

```
String name;
name = JOptionPane.showInputDialog(null, "Firs- name");
```

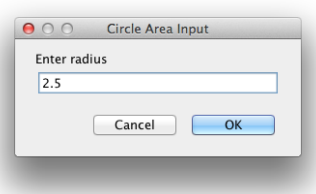


3) To convert this input to other data type

```
Double.parseDouble (string name);
Interger.parseInt (string name);
```

Example

```
String name;
name = JOptionPane.showInputDialog(null "Enter radius");
double number = Double.parseDouble(nome);
```



CHAPTER 3 DECISIONSTRUCTURE

1. 1- statmen- create a decision allowing program follow a path

2. Example

```
if (BooleanExpression) {   <-- if true Excute
    statment
}
else                       <-- false excute
    statement
```

3. Relational Operator

`>` `<` `>=` `<=` `==` `!=`

! greater, less than

4. No Semicolon at end of if statment

Example

```
if ( value >95) ;   <-- not allowed
    statment
```

5. Having Multiple statment excute use a braces

EX

```
if (value < 5) {
    value = 1;
    value = value-1;      > multiple statment
}                         <-- bracket
```

6. Example of a if-else statment

```
int value= 2;
if ( value >0) {
    System.out.println ("positive number");
else {
    System.out.println ("negative number");
}
```

7. Nested if statment

- When a if stement is inside another if statment

Example

```
int money = 3;

if ( money > 0 ) {
    if ( money > 2 ){
        statment= you can buy something;
    }
    else ( quantity == "slow") {
        statment= you cant buy something;
    }
```

```
} else {
        statment = you can't buy something;
}
```

8. If - else - if statment
   - it is a series of condition

Example

```
if ( expression) {

}
else if (expression) {

}
else if ( expression) {

}
else {

}
```

9. Logical operator

   && - and (Both boolean expression must be true for it to be true)

   || - or (One boolean expression must be true for it to be true)

   ! - not (reverse a boolean expression)

! - not (...

Example

```
String name = JOptionPane.showInputDialog("Enter number 1-10");

int number = Integer.parseInt(name);

if ((number >= 1) && (number <= 10)) {
    JOptionPane.showMessageDialog("number is within range");
}
else {
    JOptionPane.showMessageDialog("Not within range");
}
```

10. Comparing string
    - You can't use relational operator to compare string because it compare their memory address

```
StringReference1.equals(StringReference2) - this to see it equal
```

Example

```
if (name1.equal(name2)) {
    statement = it equal eachother;
}
```

...value

if (StrngReference . comapare (otherString) == ' ')

- it is used to compare to see it it equal, less, greather than
- if it is 0 it mean it equal
- if it is 1 it mean stringReferene is greater that otherString.

11. Switch statment is used to determine where the progrom branch

Ex

```
switch ( testExpression )
{
    case value_1 :
        statement;
        statment;
        break;
    case value_2 :
        statment;
        statment;
        break;

    default :
        statment;
        statment;
        break;
}
```

12. Displaying a formatted output with System.out.printf()

System.out.printf (formatString, AgrumentList);

Example

System.out.printf ("Our sale %f ", sales);

%f — format specifie.

sales — to float

System.out.printf("Our sale %f is %f big", sales, big);

13. format specifier Syntax

%[flagg] [width] [.precision] conversion

conversion: f or d (f: float or double; d: integer)

14. precision specifier
- Cause number to change number of decimal
- Example

System.out.printf("%.2f", temp);

(.2f: float or double; rounded to 2 decimal place)

15 Minimum field width
- It is minum value of space used to display valu

Example

```
double number = 23.2;
System.out.printf("%6f", number);
```

output: _ _ 2 3 . 2

6 space

16. Combing feld width and setprecision

```
double number = 23.2234
System.out.printf("%7.2f", number);
```

output: _ _ 2 3 . 2 2

width 7 and 2 decimal place

17. Flag
- Cause number to be formatted in specific way
  - Display number with comma seperation
  - To left-justify numbe
  - To pad number with leading zero

Comma seperation

```
double amount = 122345.23
System.out.printf("%,f", amount);
```

output: 122,345.23

padding with leading zero

```
int number= 1234;
System.out.printf("%05d", number);
```

output: 01234

left justify number

Add a minus sign

Example

```
int num1 = 123;
int num2 = 12;
System.out.printf("%-4d %-4d", num1,
                                num2);
```

(%-4d: integer value)

18. formating a string argument

a) field width

```
String name = "Rain";
System.out.printf("%6s", name)
```

output: __Rain

6 width

b) left justify string

```
String name1= "George";
String name 2 = "Saran";
String name3 = "Jay";
String name4 = "Ozzy";
System.out.printf("%-10s%-10s\n", name1,
                                  name2);
```

```
System.out.printf("%-10s%-10s\n", name3, name4);
```

Output: George_____Franklin__
Jay_____Ozzy_____

CHAPTER 4 LOOP AND FILES

1. Prefix and posfix increment and decreament
   - prefix cause the statment to increment or decrement first while posfix cause it to increment and decrement last

Example

```
int num = 5;
System.out.println(++num);   // cause it to happen first
System.out.println(num--);   // display value then decrement
System.out.println(num);
```

output:

```
6
6
5
```

2. While loop
   - it cause a group of statment to repeat if true

Example

```
while( number <= 5 ){
    System.out.println("Hello");
    number++;
}
```

output:

```
Hello
Hello
Hello
Hello
Hello
```

3. While loop is a pretested loop which test before each iteration.
4. while loop can be use for input validation

Example

```
System.out.println("Enter a number for 1 to 100");
num = keyboard.nextInt();
while((num < 1) || (num > 100)){
```

```
        System.out.println("Enter a valid number");
        num = keyboard.nextInt();
    }
```

5. Do while loop
   - It is a posttest loop that is tested after first iteration

Example

```
do {
        statment;
        System.out.println("Enter y for yes and N for No");
        input = keyboard.nextLine();
        repeat = input.charAt(0);

    } while (repeat == 'Y' || repeat == 'y');
```

6. Use the do-while loop when you want the exution to excute ones.

7. For loop
   - used best when we know amount of iteration

Example

```
for( intilization; Test; Update){
}
```

8. Example of for loop

```
for (int i =0 ; count<=5 ; count++ ){
    System.out.println("Hello");
}
```

output:

```
Hello
Hello
Hello
Hello
Hello
```

9. Using multiple statment in the intilizatio.

```
int x, y;
for( x=1, y=1; x<=5; x++, y++){
        System.out.println(x + " plus " + y + " equal " +(x+y));
}
```

output

1 plus 1 equal 2
2 plus 2 equl 4
3 plus 3 equal 6
4 plus 4 equal 8
5 plus 5 equal 10

10. Running total and Sentimental value

- it is used to accumulate a sum of value

Example

```
totalSale = 0;
for (int count = 0; count < 5; count++)
      System.out.println("Enter the sale" + count);
      sales = keyboard.nextInt();
      totalSale += sales;
}
```

↑ accumulate sum

11. Nested loop is a loop inside a loop

```
for (int hours = 1; hours <= 12; hours++)
{
    for (int minutes = 0; minutes <= 59; minutes++)
    {
       for (int seconds = 0; seconds <= 59; seconds++)
```

[redacted]

```
       {
          System.out.printf("%02d:%02d:%02d\n", hours, minutes,
       }
    }
}
```

Output :

```
01:00:00
01:00:01
01:00:02
01:00:03

(The loop continues to count . . . )

12:59:57
12:59:58
12:59:59
```

12. Break and Continues Statment

- Break cause the loop to terminate
- Continue, it to stop current iteration and start new

Example

```
for ( int i=0; i<5 ; i++ {
        if ( i == 2 ) {
            break;
        }
    System.out.println( i );
}
```

output :

0
1

Example

```
for ( int i=0; i<5 ; i++ {
        if ( i == 2 ) {
            continue;
        }
    System.out.println( i );
}
```

Output :

0
1
3
4
5

13) Deciding which loop to use

- While loop: used when we don't know amount of iteration. Also ideal for input validation.
- do-while loop: used when we want to iterate at least once.
- for-loop: used when we know the amount of iteration.

14) File Input and Output

File used by a program

i) Opened
ii) written or read from file
iii) close file

15) Java provide an API to work with file

```
import java.io.*;
```

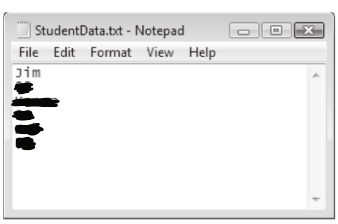
16) Writing to a file which never exist

OPEN — `PrintWriter outputfile = new PrintWriter("StudentData.txt");`
↑ open the file you want to write to

WRITE — `outputfile.println("Jim");`

CLOSE — `outputfile.close();`

output:



17) Throw Clause to Method header

- It is used when a program throw an exception which show no space when trying to create a file. This tell you what type of exception. Always use when using PrintWriter Object.

Example

```
public static void main (String[] args) throws IOException
{

}
```

18. Writing to an existing file

Allow you to write to existing file

```
OPEN  - FileWriter fwrite = new FileWriter("MyFriend.txt", true);
        PrintWrter ouputfile = new PrintWriter(fwrite);

WRITE - outputFile.println("Bill");
CLOSE - outputFile.close();
```

19. Reading from file

```
OPEN:  File file = new File (filename);
READ:  Scanner inputfile = ew Scanner(file);
       String name = inputFile.nextLine();
       System.out.println(name);
CLOSE: inputfile.close();
```

20. Reading from a file to end of file

```
File file = new File (filename);
Scanner inputfile = new Scanner (file);
while (inputFile.hasNext()) {        read to end of file
    String friendName = inputFile.nexLine();
    System.out.println(friendName);
}
```

21. Checking for file existence

```
File file = new File("Number.txt");
if (!file.exist()) {        check for file existence
```

```
        System.out.println("The file doesn't exist");
        System.exit(0);
    }
```

finish the program if the file doesn't exist

22. Generating Random number

```
import java.util.Random;
Random randomNumber = new Random();
number = randomNumber.nextInt();
```

← create Random Object

↑ create a random Integer

```
number = randomNumber.nextInt(10) + 1;
```

↑ random number between 1 - 10

↑ starting value

```
number = randomNumber.nextInt(100) - 50;
```

between -50 and -49

```
number = randomNumber.nextInt(10);
```

between: 0 - 9

CHAPTER 5 METHODS

1. Method break program to small excutable Program
   - void method
   - value returing method

2. We used method is the main which is predefined

3. Example showing how method is devided

```
public static void main(String[] args) {
    statment;
    statment;
}
Public static void method2( ) {
    Statment;
    statment;
}
```

main method

method 2

3. Two type of method return type
   a) Void method : Perform task and terminate

Example

```
public static void displayMessage1( ){
    System.out.println("Hello World");
}
```

return type

- Public static — mean it available to code outside the class
- void - it is the return type
- displayMessage - is the method name
- ( ) - accept argument

4. How a method is excuted
   - it excute by calling the method name.

Example

displayMessge( );

↑ This excute the method

5.

```
public class SimpleMethod
{
   public static void main(String[] args)
   {
      System.out.println("Hello from the main method.");
      displayMessage();
      System.out.println("Back in the main method.");
   }

   /**
      The displayMessage method displays a greeting.
   */

   public static void displayMessage()
   {
      System.out.println("Hello from the displayMessage method.");
   }
}
```

This call the method

Method header

Output:

Hello from the main method.
Hello from the displayMessage Method.
Back in the main method.

6)

```
6
7  public class CreditCard
8  {
9     public static void main(String[] args)
0     {
1        double salary;       // Annual salary
2        int creditRating;    // Credit rating
3        String input;        // To hold the user's input
4
5        // Get the user's annual salary.
6        input = JOptionPane.showInputDialog("What is " +
7                                    "your annual salary?");
8        salary = Double.parseDouble(input);
9
0        // Get the user's credit rating (1 through 10).
1        input = JOptionPane.showInputDialog("On a scale of " +
2              "1 through 10, what is your credit rating?\n" +
3              "(10 = excellent, 1 = very bad)");
4        creditRating = Integer.parseInt(input);
5
6        // Determine whether the user qualifies.
7        if (salary >= 20000 && creditRating >= 7)
8           qualify();
9        else
0           noQualify();
1
2        System.exit(0);
3     }
4
```

Input:

3000
8

Output:

Congratulation you qualify for credit card!

no new line

```
   public static void qualify()
   {
      JOptionPane.showMessageDialog(null, "Congratulations! " +
                                 "You qualify for the credit card!");
   }

   /**
      The noQualify method informs the user that he
      or she does not qualify for the credit card.
   */

   public static void noQualify()
   {
      JOptionPane.showMessageDialog(null, "I'm sorry. You " +
                           "do not qualify for the credit card.");
   }
}
```

7 Hierarchical Method call

```
public class DeepAndDeeper
{
   public static void main(String[] args)
   {
      System.out.println("I am starting in main.");
      deep();
      System.out.println("Now I am back in main.");
   }

   /**
      The deep method displays a message and then calls
      the deeper method.
   */

   public static void deep()
   {
      System.out.println("I am now in deep.");
      deeper();
      System.out.println("Now I am back in deep.");
   }

   /**
      The deeper method simply displays a message.
   */

   public static void deeper()
   {
      System.out.println("I am now in deeper.");
   }
}
```

Output:

I am starting in main
I am in deep
I am now in deeper
Now I am back in deep
Now I am back in main.

8. Passing argument to Method

```
public static void main (String[] args){
        displayValue (5);
}
public static void displayValue (int value){
        System.out.println (value);
}
```

argument copied to it.

parameter

9. Incorrect passing of argument

```
displayValue (int x);   WRONG
displayValue (5);       RIGHT
```

10. Whe passing a value make sure the data type is compatable

```
a) public static void main (String[] args){
       displayValue (1.5);
   }
   public static void displayValue (int value)
```

- it wrong because it not compatable
- The right form is double value

11. No statment outside the parameter can access variable

```
public static void main (String[] args){
    None (5);
    System.out.println (value);
}
public static void None (value) {
}
```

can't access parameter variable

it is WRONG

12. Passing multiple item to paremter

```
public static void main (String[] args){
    displayValue (1.5, 2.5);
}
public static vod None (double value1, double Value2)
}
```

passing multiple item to paronetes

13 Argument passed by value

- It mean it just copies the value to parameter.

Any change made to the parameter value never after the variable pass in.

Example

```
public static void main (String[] args){
    double value1 = 2.5;
    double value2 = 3.5;
     displayValue (value1, value2);
     System.out.println ("the value1 in main is " + value1);
}
Pubic static vod None (double value1, double value2 ){
      value1 = 0.0;
      System.out.print ("The value1 in None is " + value1);
}
```

output:
The value1 in None is 0.0
The value1 in main is 2.5.

14. Passing object Reference to Method

```
public static void main(String[] args ) {
      String name = "Sarah";
       System.out.println (name);
      Try (name);
       System.out.println (name);
}
public static void Try( String str ) {
      str = "Dicken";
      System.out.println(str);
}
```

pass object Reference

this str point to a new memory address which never change variable in the call function

output:
Sarah
Dicken
Sarah

15. @ param tag
- it is used for documentation of the parameter

Example

```
/**
   The showSum meth display the sum of two numbe
   @ param num1 the first number
   @ param num2 the second number
*/
public static void showSum (double num1, double num2)
{

}
```

Documentation for showSum

showSum

```
public static void showSum(double num1,
                           double num2)
```

The showSum method displays the sum of two numbers.

Parameters:
num1 - The first number.
num2 - The second number.

16.) Local Variable
- Local variable declared inside a method is not accesible to statment outside methode

```
public static void texas()
{
   int birds = 5000;

   System.out.println("In texas there are " +
                      birds + " birds.");
}

/**
   The california method also has a local variable named birds.
*/
```

← can see one at a time

```
r 5  Methods

public static void california()
{
   int birds = 3500;

   System.out.println("In california there are " +
                      birds + " birds.");
}
```

← can see one at a time which make the variable local

- Local variable only exit when the method run

17. Returning a value from method
    - It send value back to the Statment that called the method

Example

```
public class Main {
    public static void main (String[] a-g s) {
        int value = 2;
        int value2 = 3;
        int sum = SumValue (value, value2);
        System.out.println ( sum);
    }
    public static int SumValue ( int value1, int value2){
        int sum = value1 + value2;
        return sum;
    }
}
```

The returned value

return value int

18. @return Description ← Documentation of return

19. Returning a boolean type.

```
public static boolean isValid (int number) {
    boolean statues;
    if ( number >= 1 && number <= 100 )
        statue = true;
    else statue = false;

    return statues;
```

$\int_{\phi}$

CHAPTER 6 Object and Classes

1. An object something that serve a specific purpose
   - An object is created from class

   Example
   A class serve as blueprint for a house while the object build serveral house

2. An object
   - store data (field)
   - perform a task with the method

3. Before we can use a class we have created an object of the class in the memory.

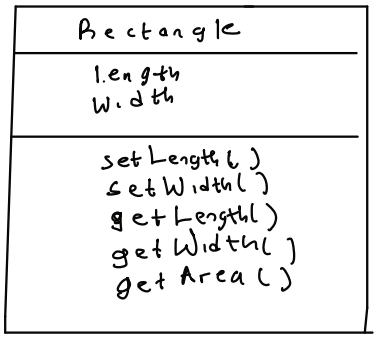
4. Just like when we create a Scanner object we are creating instance of scanner class

5. When working with object we create a memory address that reference the object by a variable.

   Example
   ```
   Random rand = new Random();
   ```
   ↑ variable reference the Random object

6. Writing a simple class for Rectangle
   a) Length : hold the rectangl length
   width : hold rectangle width
   SetLength : hold method that set value of length
   SetWidth : hold method that set value of width
   getLength : return value of length
   getWidth : return value of width
   getArea : return are of retangle.

7 UML diagram showing rectangle class



8. Code for a class
AccessSpecifier class Name {
    Members
}

Example

public class Rectangle {
    Members
}

9. The public mean code outside the class can access it

10. Writing the class field

```
public class Rectangle {
    private double length;
    private double width;
}
```

The keyword private means it is hidden from code outside class.

11. When it is hidden it prevented from corruption. Only the public method have access to those field.

12. Rec.java

```
public class Rectangle {
    private double length;
    private double width;
    public void setLength(double len) {
        length = len;
```

it doesn't return any value

the setlength set the length

```
}                                                    of variable
public void setWidth (double wid) {
        width = wid;
}
}
public class LengthDemo {
public static void main(String[] args) {
        Rectangle box = new Rectangle();
                        ↑ this create an object in the
                          memory
        box.setLength(10);
        box.setWidth(5)
                        ↑ send the value to
                          setLength in Rect.jav
}                       ↑ send value to
                          setWidth in Rect.java
}
```

13.

14. Writing the code that getLength and getWidth

```
public class Rectangle {
        private double length;
        private double width;
        public void setLength(double len) {
                length = len;
        }
        public void setWidth (double wid) {
                width = wid;
        }
        public double getLength( ) {
                return length;
        }
        public double getWidth( ) {
        }
        public double area( ) {
                return length * width;
        }
}
public class Demo {
public static void main (String[] args){
        Rectangle rec = new Rectangle( );
```

```
rec.setLength(5.0);
rec.setWidth(2.5);
System.out.println("length is"+rec.getLength());
System.out.println("width is"+rec.getWidth());
System.out.println("Area is"+rec.getArea());
```

9

```
}
}
```

Output:

```
5.0
2.5
12.50
```

15. A method that get a value from a class field is called accessor

example

```
rec.getLength();
```

16. A method that store a value in a field is called mutator

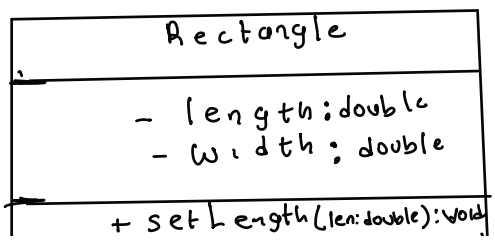
example

```
rec.setLength(5);
```

17. The reason for data hiding is to
- Used in a software component to prevent the change of the data

18. Avoiding stale data
- The reason area is not stored in field is because the value depended on other value which makes it not to updated.

19. Showing Access specifier in UML Diagram



+ setWidth(w: double): void
+ getLength( ): double
+ getWidth( ): double
+ getArea( ): double

20. A class could have various instance

```
public class Demo {
    public static void main (String[] args) {
        Rectangle rec1 = new Rectangle();
        Rectangle rec2 = new Rectangle();
    }
}
```

different instance of class

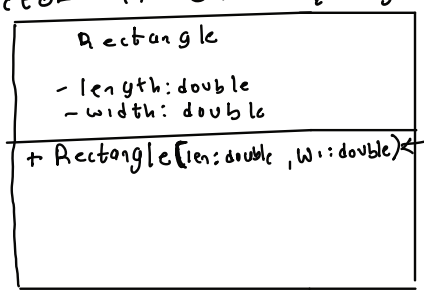
21. Constructor
- It perform the intial intilization of a field in a class

Example

```
public class Rectangle{
    private double length;
    private double width;
    public Rectangle (double len, double wid ){
        length = len;
        width = wid;
    }
}
public class Demo {
    public static void main(String[] args ) {
        Rectangle rec = new Rectangle(8, 5);
    }
}
```

↑ pass it to the constructor

22. Constructor in UML Diagram



23. Default Constructor
- it is automatically created when we don't create one which set all the field variable to 0.

Example

```
Class Rectangle {
    private length;  <-- set to 0 because of default constructor
}
Rectangle rec = new Rectangle();  <-- default constructor
```

24. No arg Constructor
- is a Constructor that never accept value

```
public Rectangle ( ) {        <-- it never accept value
    width = 1.0;
    length = 2.0;
}

Rectangle rec = new Rectangle();
                <-- calls the constructor
```

25. Creating a String Object
- We can use this because it common, that why

```
String name = "Ja";
```

- Or we can create a Object

```
String name = new String("Joe");
```

26. Passing object as argument to method

Example

```
void showDieSides(Die d) {
    System.out.println("This die has "
        + d.getSide() +
        "sides.");
```

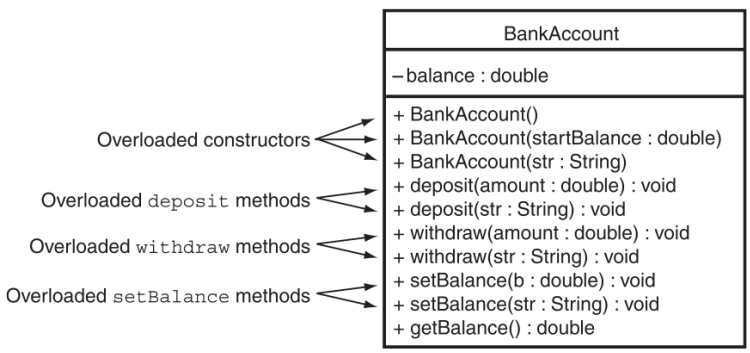
27. Overloading Constructor
- When two constructor have same name but different parameter

Ex

```
class Rectangle {
    public Rectangle ( ) {
        length = 0.0;
        Width = 0.0;
    }
    public Rectangle ( double len , double wid ){
        length = len;
        width = wid;
    }
}
public static void main(String[ ] args){
    Rectangle rec1 = new Rectangle( );
    Rectangle box2 = new Rectangle (50, 10.0);
}
```

this don't accept argument

28) Bank Account UML Diagram



2.47)

```
public class BankAccount
{
   private double balance;      // Account balance

   /**
      This constructor sets the starting balance
      at 0.0.
   */

   public BankAccount()
   {
      balance = 0.0;
   }

   /**
      This constructor sets the starting balance
      to the value passed as an argument.
      @param startBalance The starting balance.
   */

   public BankAccount(double startBalance)
   {
      balance = startBalance;
   }

   /**
      This constructor sets the starting balance
      to the value in the String argument.
      @param str The starting balance, as a String.
   */

   public BankAccount(String str)
   {
      balance = Double.parseDouble(str);
   }



    public void deposit(double amount)
    {
       balance += amount;
    }

    /**
       The deposit method makes a deposit into
       the account.
       @param str The amount to add to the
                  balance field, as a String.
    */

    public void deposit(String str)
    {
       balance += Double.parseDouble(str);
    }

    /**
       The withdraw method withdraws an amount
       from the account.
       @param amount The amount to subtract from
                     the balance field.
    */

    public void withdraw(double amount)
    {
       balance -= amount;
    }

    /**
       The withdraw method withdraws an amount
       from the account.
       @param str The amount to subtract from
                  the balance field, as a String.
    */

    public void withdraw(String str)
    {
       balance -= Double.parseDouble(str);
    }
```

```
   public void setBalance(double b)
   {
      balance = b;
   }

   /**
      The setBalance method sets the account balance.
      @param str The value, as a String, to store in
                 the balance field.
   */

   public void setBalance(String str)
   {
      balance = Double.parseDouble(str);
    }

   /**
      The getBalance method returns the
      account balance.
      @return The value in the balance field.
   */

   public double getBalance()
   {
      return balance;
   }
}
```

```
public class AccountTest
{
   public static void main(String[] args)
   {
      String input;    // To hold user input

      // Get the starting balance.
      input = JOptionPane.showInputDialog(
                "What is your account's starting balance?");

      // Create a BankAccount object.
      BankAccount account = new BankAccount(input);

      // Get the amount of pay.
      input = JOptionPane.showInputDialog(
                "How much were you paid this month?");

      // Deposit the user's pay into the account.
      account.deposit(input);

      // Display the new balance.
      JOptionPane.showMessageDialog(null,
         String.format("Your pay has been deposited.\n" +
                       "Your current balance is $%,.2f",
                       account.getBalance()));

      // Withdraw some cash from the account.
      input = JOptionPane.showInputDialog(
                "How much would you like to withdraw?");
      account.withdraw(input);

      // Display the new balance
      JOptionPane.showMessageDialog(null,
         String.format("Now your balance is $%,.2f",
                       account.getBalance()));

      System.exit(0);
   }
}
```

30. Scope of intance field
- instance field private is visible to only the class intance method

31. while instance field defined public can be access by code outside class

32. We can have a local variable and parameter variable with the same name

33. Object Oriented Design Analysis

a) identify class and object

a) We know class consist of data and function
- so we think of those that require it

b) Also we could use this as candidate

i) menu

ii) Physical Object (Vechicle, machine)

iii) Record keeping item

a) payroll

b) customer historie

b) Define each class Attribute

a) Data element of class

Example

Class name: MenuIten

Attribute: price
category

c) Define each class behavior

a) the program activities

i) change value

ii) display pri

d) Define Relationship

a) The relationship between the private and public

34) Finding a class

a) Identify all Noun

i) Noun

ii) Phrases

iii) pronoun

b) Refine the Noun

- Some mean some thing
- Be nothing

c) Remove some that we should not be concerned about

d Some might represent Object remove

CHAPTER 7 Array and ArrayList Class

1. Declaring an array in Java

```
int[] number = new int[6];
```

size of the array

↑ The bracket indicate this variable is reference of int array

2.
```
final int NUM_ELEMENT = 6;
int[] number = new int[NUM_ELEMENT];
```

↑ Always use final for number of element

3. Once array is created the size can't change

4. Accessing element of array

```
number[0] = 20;
number[1] = 10;
```

access index 0 of the array

↑ Access index 1 of the array

5. By defaut Java intilize array element with 0

6. Accessing and display element of array

```
*/

public class ArrayDemo2
{
   public static void main(String[] args)
   {
      final int EMPLOYEES = 3;               // Number of employees
      int[] hours = new int[EMPLOYEES];      // Array of hours

      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      System.out.println("Enter the hours worked by " +
                         EMPLOYEES + " employees.");

      // Get the hours for each employee.
      for (int index = 0; index < EMPLOYEES; index++)
      {
         System.out.print("Employee " + (index + 1) + ": ");
         hours[index] = keyboard.nextInt();
      }

      System.out.println("The hours you entered are:");

      // Display the values entered.
      for (int index = 0; index < EMPLOYEES; index++)
         System.out.println(hours[index]);
   }
}
```

Array

intilizing array

displaying what in an array

Progr Output:

Enter the hours worked by 3 employee
Employee 1:
40 [Enter]

Employee 2:
20 [Enter]

Employee 3:
15 [Enter]

7. Array in Java perform bound checking of array to prevent of of bound input

Example

```
int[] num = new int[2];
num[0] = 2;
num[1] = 3;
num[2] = 4;
```

why is because it has only two element

Output

Out of bound

8. Array start at index 0.

9. Array initilization

```
int[] array = {1, 2, 3}
or
int array[] = {1, 2, 3}
```

10. Declaring arrays intilization

```
int number[], code[], scores[];
```

11 Pre increment or decrement of array

```
++num[2];   correct
num[2--];   is wrong because it change value of subscript
```

12. TO get the size of an array

Ex

```
double[] temperature = new double[25];
size = temperature.length;
```

Output

25

13 Enhanced for loop runs through an array

Example

```
int[] number = {3, 2, 5}
for(int value : number) {
    System.out.println(val);
}
```

output

3
2
5

14. We can reassign an array reference variable

Ex

```
int[] number = new int[10];
number = new int[5];
```

↖ reassign the variable address

15. Copying array

Ex

```
int[] array1 = {2, 3, 4};
int[] array2 = array1;
```

↑ This doesn't copy the array it assign the array address to array2 which causes both array1 and array2 to point to same address.

```
{
   public static void main(String[] args)
   {
      int[] array1 = { 2, 4, 6, 8, 10 };
      int[] array2 = array1;

      // Change one of the elements using array1.
      array1[0] = 200;

      // Change one of the elements using array2.
      array2[4] = 1000;

      // Display all the elements using array1
      System.out.println("The contents of array1:");
      for (int value : array1)
         System.out.print(value + " ");
      System.out.println();

      // Display all the elements using array2
      System.out.println("The contents of array2:");
      for (int value : array2)
         System.out.print(value + " ");
      System.out.println();
   }
```

Output:

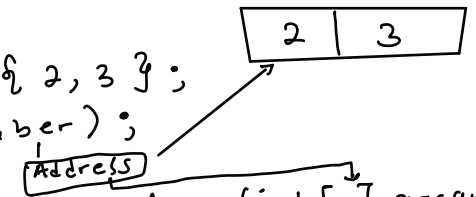
The content of array1 :
200 4 6 8 1000
The content of array2:
200 4 6 8 1000

16. Passing array to a method

- When you pass an array to a method you are simply passing the value that reference orginal array

Ex

```
int [] number = { 2, 3 };
showArray(number);
```




```
public static void showArray(int [] array){
```

17. You can't compare array with this

```
int [] array1 = { 5, 10, 15 };
int [] array2 = { 4, 8, 12 };
if ( array1 == array2 )
    System.out.println ("The array are the same");
```

← This compare the array address of variable

18. TO compare arrays

```
int[] firstArray = { 2, 4, 6, 8, 10 };
int[] secondArray = { 2, 4, 6, 8, 10 };
boolean arraysEqual = true;   // Flag variable
int index = 0;                // Loop control variable

// First determine whether the arrays are the same size.
if (firstArray.length != secondArray.length)
   arraysEqual = false;

// Next determine whether the elements contain the same data.
while (arraysEqual && index < firstArray.length)
{
   if (firstArray[index] != secondArray[index])
      arraysEqual = false;
   index++;
}

if (arraysEqual)
   System.out.println("The arrays are equal.");
else
   System.out.println("The arrays are not equal.");
```

19. Summing value of numeric array

```
int[]unit = new int[ 5 ];
for (int i =0 ; i < unit.length ; i++ ) {
        unit [ i ] = keyboard.nextInt();
}
int total = 0;
for ( int index =0 ; index < unit.length ; index++ ) {
        total += unit[index];
```

```
}
```

Output

20. Working with array and Files

Writing to a file with an array

Ex

```
int [ ] number = { 10, 20, 30 };
PrintWriter outputfile = new PrintWriter("value.txt");
for (int index=0; index< number.length ;index) {
        outputfile.println(number[index]);
}
outputfile.close();
```

21. Returning array from method

this return reference to the array

Ex

```
public static double[ ] array( ) {
    double[ ] array = { 1, 2, 3 };
    return array;
}
```

```
public static void main(String[] args)
{
   double[] values;

   values = getArray();
   for (double num : values)
      System.out.print(num + " ");
}

/**
   getArray method
   @return A reference to an array of doubles.
*/

public static double[] getArray()
{
   double[] array = { 1.2, 2.3, 4.5, 6.7, 8.9 };

   return array;

}
```

22. An array of string object

- It is a reference to a array of string object

Example

```
String[ ] name = { "Bill", "Sarah" };
```

or

```
String[ ] name = new String[2];
name[0] = "Bill";
name[1] = "Sarah";
```

23. calling String method for array string object

Ex

```
System.out.println(name[0].toUpperCase());
```

↑ This calles the method of a string

24. Array of object

Example

```
BankAccount[ ] accounts = new BankAccount[5];
```

- This is an array of object of instorce of class

25. Each element of the array is intilize to null because they do not yet reference any object.

- This code reference them

```
for(int index=0; index < accounts.length; index++){
    account[index]= new BankAccount();
}
```

- No argument constructor assign 0 to the balance field of class

2.16.

```
public class ObjectArray
{
   public static void main(String[] args)
   {
      final int NUM_ACCOUNTS = 3;  // Number of accounts

      // Create an array that can reference
      // BankAccount objects.
      BankAccount[] accounts = new BankAccount[NUM_ACCOUNTS];

      // Create objects for the array.
      createAccounts(accounts);

      // Display the balances of each account.
      System.out.println("Here are the balances " +
                         "for each account:");

      for (int index = 0; index < accounts.length; index++)
      {
         System.out.println("Account " + (index + 1) +
                    ": $" + accounts[index].getBalance());
      }
   }

   /**
      The createAccounts method creates a BankAccount
      object for each element of an array. The user
      is asked for each account's balance.
      @param array The array to reference the accounts
   */

   private static void createAccounts(BankAccount[] array)
   {
      double balance;   // To hold an account balance

      // Create a Scanner object.
      Scanner keyboard = new Scanner(System.in);



         // Create the accounts.
         for (int index = 0; index < array.length; index++)
         {
            // Get the account's balance.
            System.out.print("Enter the balance for " +
                             "account " + (index + 1) + ": ");
            balance = keyboard.nextDouble();

            // Create the account.
            array[index] = new BankAccount(balance);
         }
      }
   }
```

This create object of object for the array

27. The sequential search algorithm
- it is used to search for a specific value in an array

```
public class SearchArray
{
   public static void main(String[] args)
   {
      int[] tests = { 87, 75, 98, 100, 82 };
      int results;

      // Search the array for the value 100.
      results = sequentialSearch(tests, 100);

      // Determine whether 100 was found and
      // display an appropriate message.
      if (results == -1)
      {
         System.out.println("You did not " +
                            "earn 100 on any test.");
      }
      else
      {
         System.out.println("You earned 100 " +
                            "on test " + (results + 1));
      }
   }

   /**
      The sequentialSearch method searches an array for
      a value.
      @param array The array to search.
      @param value The value to search for.
      @return The subscript of the value if found in the
              array, otherwise -1.
   */



   public static int sequentialSearch(int[] array,
                                      int value)
   {
      int index;         // Loop control variable
      int element;       // Element the value is found at
      boolean found;     // Flag indicating search results

      // Element 0 is the starting point of the search.
      index = 0;

      // Store the default values element and found.
      element = -1;
      found = false;

      // Search the array.
      while (!found && index < array.length)
      {
         if (array[index] == value)
         {
            found = true;
            element = index;
         }
         index++;
      }

      return element;
   }
}
```

28. Two dimesional Array

- it haw row and columns

Example

	Column 0	Column 1	colum 2
Row 0	score[0][0]	score[0][1]	score[0][2]
Row 1	score[1][0]	score[1][1]	score[1][2]
Row 2	score[2][0]	score[2][1]	score[2][2]

```
double[][] scores = new double[3][3]
```

↑ this define a two dimesional array

29. How to access element of two dimesional array

scores[0][0] — it access row 0 and column 0

Scores[0][1] — it access row 0 and column 1

30. Usany nested for loop to process two dimesional array

Ex

```
final int ROWS = 3;
final int COLOUM = 4;
double[][] score = new double[ROWS][COLS];
for(int row=0; row<ROWS;row++){
      for(int col=0; col<COLOUM; col++){
            System.out.println("Enter a score: ");
             scores[row][col] =Keyboard.nextDouble();
       }
}
```

31) initilizing a two dimesional array

Ex

```
int[][] numbers = {{1,2,3}  {4,5,6}};
```

or

```
int [ ][ ] numbers = { { 1, 2, 3 },
                       { 4, 5, 6 } }; ← This way for more clarity
```

32) length of two dimesional array

```
int [ ][ ] number = { { 1, 2, 3 }
                      { 2, 3, 4 } };

for ( int row,=0 . row < number . length ; row++ ){
    for (int col=0 ; col < number [row] . length; col++){
            System.out . println ( number[row] [cols];
    }
}
```

33) Summing all element of a two-dmesional array

```
int [ ][ ] number = { { 1, 2, 3 }
                      { 2, 3, 4 } };

int total=0;
for (int row=0 ; row < number.length ; row++ ) {
    for ( int col=0; col < number [row] . length; col++){
            total += number [row] [col ];
    }
}
  System.out.println ("The total is " + total );
```

34) Passing a two dimesional to method

```
public class Main {
    public static void main ( String[ ] arg ) {
        int [ ] [ ] number = { { 1, 2, 3 }
```

{ 2, 5, 4 } };

showArray(number);

reference to a two dimesional array

}

}

public static void showArray(int[][] array) {

}

35. Ragged Array

When a two dimesional has different length of row it is called ragged array.

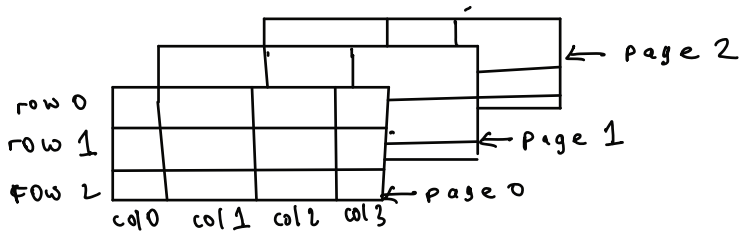
Ex

	colum 0	colum 1	colum 2	colum 3
row 0	*	*		
row 1	*	*	*	
row 2	*	*	*	*

row 0 has 2 colum
row 1 has 3 colum
row 2 has 4 colum

```
int [][] ragged = new int [4][];
ragged[0] = new int [2];
ragged[1] = new int [3];
ragged[2] = new int [4];
```

36. Array with three dimesion

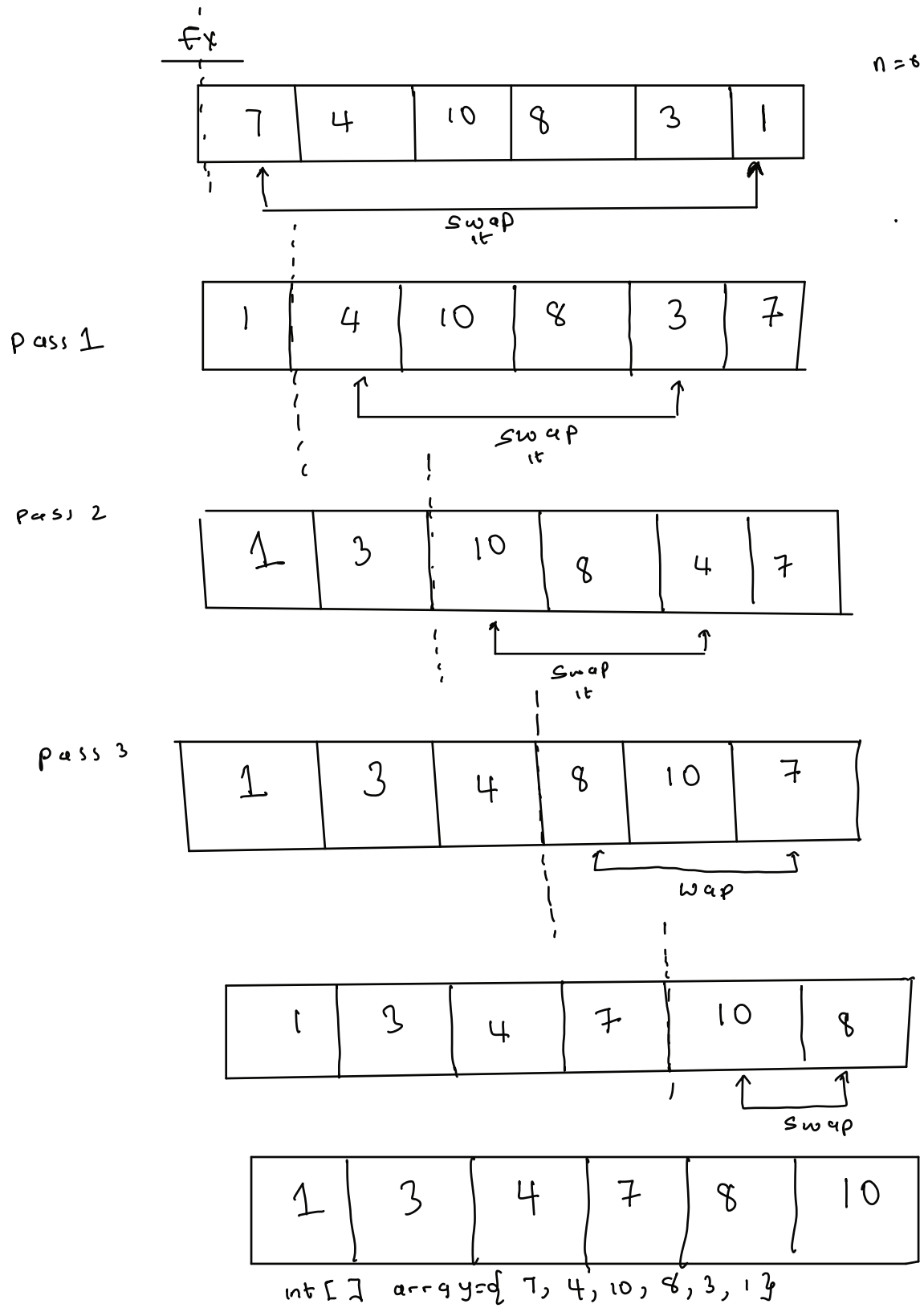



```
double[][][] seat = new double[3][3][4];
```

page, row, colum

37. Selection sort

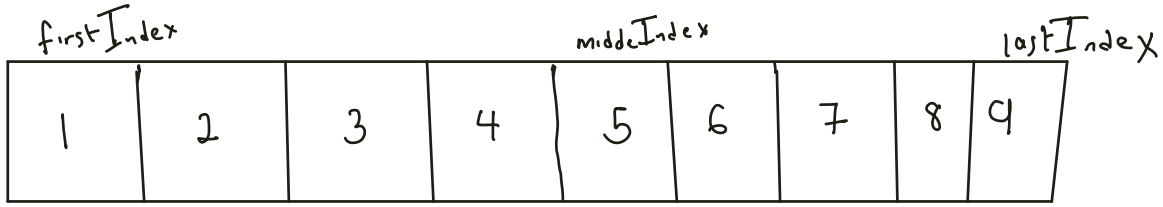
- It is the arrangement of data in order



int [] arr = { 7, 4, 10, 8, 3, 1 }

```
int i, j, minValue, minIndex = 0;
for (int i = 0; i < array.length-1; i++){
        minIndex = i;
        minValue = array[minIndex];
       for(int j = i+1; j < array.length; j++){
            if(array[j] < minValue) {
                minIndex = j;
                minValue = array[minIndex];
            }
       }
       if (minValue != array[i]){
            temp = array[i];
            array[i] = minValue;
            array[minIndex] = temp;
       }
}
```

38. Binary search Algorithm




```
int firstIndex = 0;
int lastIndex = arr.length;
int middleIndex;
int position = -1;
int found = false;
int value = 2;
while (!found && firstIndex <= lastIndex){
        middleIndex = (firstIndex + lastIndex)/2;
        if (value == array[middleIndex]){
```

```
            position = middleIndex;
            found = true;
        }
        else if ( value < array[middleIndex] ) {
            lastIndex = middleIndex - 1;
        }
        else {
            firstIndex = middleIndex + 1;
        }
    }
    if ( position == -1 ) {
        System.out.println ("The index of value not found");
    else {
        System.out.println ("The Index is" + position);
    }
```

39) Command line Argument

- It is thing passed from the command line to main header method

Ex

```
public class commandLine {
    public static void main(String[] args ) {
        System.out.println ( args[0] + " " + args[1]);
    }
}
```

Command Line:

```
javac commandLine.java
java commandLine Hello Iyanu
```

output:

```
Hello Iyanu
```

40. Variable length argument list

- this method take any amount of variable number of argument

Ex

```
int result = sum(firstVal, secondVal, thirdVal);

public static sum (int ... numbers) {
        int total = 0;
        for(int val : numbers) {
                total += val;
        }
        return total;
}
```

This take any amount of variable number pass by the call

Ex

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

                double total;
                BankAccount account1 = new BankAccount(100.0);
                BankAccount account2 = new BankAccount(200.0);

                total = totalBalance(account1, account2);

        }
}
public static double totalBalance(BankAccount ... accounts){
        double total = 0;
        for(BankAccount acctObject : accounts) {
                total += acctObject.getBalance();
        }
        return total;
```

}

Program output

300

41) ArrayList Class

- it is a java API to similar to array that allow you to store object. it size can adjust to accomodate a number or shrink.

42) How to create an arrayList Object

```
ArrayList<String> nameList = new ArrayList<String>();
```

- import java.util.ArrayList

43) Adding to ArrayList Object

```
nameList.add("Jone");
nameList.add("Sarah");
```

44) Display item in ArrayList Object

get size of arrayList object

```
for(int index=0; index<namesList.size(); index++){
    System.out.println(namesList.get(index));
}
```

45) Example

```
public class chapter7 {
    public static void main(String[] args) {
        ArrayList<String> nameList = new ArrayList<String>();
        nameList.add("Victor");
        nameList.add("Sarah");
        nameList.add("Kone");
        for(int index=0; index<nameList.size(); index++){
            System.out.println(nameList.get(index));
        }
    }
}
```

Create an arrayList Object

add to arrayList Object

display an arrayList Object

}

46) Using enhanced for loop

Example

```
public class chapter7 {
    public static void main(String[] args) {
        ArrayList<String> nameList = new ArrayList<String>();
        nameList.add("Victor");
        nameList.add("Sarah");
        nameList.add("Kane");
        for (int value : nameList) {
            System.out.println(value);
        }
    }
}
```

Create an arrayList Object

add to arrayList Object

display an ArrayList Object

47) Returning all item of arrayList

```
ArrayList<String> nameList = new ArrayList<String>();
nameList.add("Victor");
nameList.add("Sarah");
nameList.add("Kane");
System.out.println(nameList);
```

output

[Victor, Sarah, Kane]

48) Removing an item from ArrayList

```
ArrayList<String> nameList = new ArrayList<String>();
nameList.add("Victor");
nameList.add("Sarah");
nameList.add("Kane");
System.out.println(nameList);
nameList.remove(1);
```

```
System.out.println(nameList);
```

Output

```
[Victor, Sarah, Kane]
[Victor, Kane]
```

49) Replacing item in a ArrayList

```
ArrayList<String> nameList = new ArrayList<String>();
nameList.add("Victor");
nameList.add("Sarah");
nameList.add("Kane");
System.out.println(nameList);
nameList.remove(1);
System.out.println(nameList);
nameList.set(1,"Becky");
System.out.println(nameList);
```

Output

```
[Victor, Sarah, Kane]
[Victor, Kane]
[Victor, Becky]
```

50) Capacity of the arrayList Object

- At first the no arg constructor hold just 10 item

```
ArrayList<String> list = new ArrayList<String>();
```

↑ just 10 item

- We can increase the capacity by

```
ArrayList<String> list = new ArrayList<String>(20);
```

↑ hold 20 item.

51) An ArrayList Object can hold different object

Example

```
ArrayList<String> list = new ArrayList<String>(20);
ArrayList<int> list = new ArrayList<int>(20);
```

52) Using the ArrayList Object to hold object

Ex

```
ArrayList<BankAccount> list = new ArrayList<BankAccount>();
list.add(new BankAccount(100.0));
list.add(new BankAccount(500.0));

for(int index=0; index<list.size(); index++){
    BankAccount accont = list.get(index);
    System.out.println(accont.getBalance());
}
```

Output

```
100.0
500.0
```

Chapter 8 Second look at Class and Object

1. Static class member belong to class but not instance of classes

2. When a field is declared static there is only one copy of the field
   ↓ to all instance of the class

3. Ex

```
Public class countable {
Private static int instanceCount = 0;

Public Countable ( ) {
      instanceCount ++;
}
public int getInstanceCount () {
      return instanceCount;
}
}
Public class Demo {
      public static void main (String[ ] args) {
            int objectCount;
            Countable object1 = new Countable ();
            Countable object2 = new Countable ();
            Countable object3 = new Countable ();
            objectCount = object1.getInstanceCount ();
            System.out.println (objectCount + "instance"+
                                "s of the class were created");
      }
}
```

This is copied to all instance of class field

Output

3 instance of class were created

4. Static method
   - Not necessary to create instance of class to excute

```
4
5  public class Metric
6  {
7     /**
8        The milesToKilometers method converts a
9        distance in miles to kilometers.
10       @param m The distance in miles.
11       @return The distance in kilometers.
12    */
13
14    public static double milesToKilometers(double m)
15    {
16       return m * 1.609;
17    }
18
19    /**
20       The kilometersToMiles method converts
21       a distance in kilometers to miles.
22       @param k The distance in kilometers.
23       @return The distance in miles.
24    */
25
26    public static double kilometersToMiles(double k)
27    {
28       return k / 1.609;
29    }
30 }
```

```
public class MetricDemo
{
   public static void main(String[] args)
   {
      String input; // To hold input
      double miles; // A distance in miles
      double kilos; // A distance in kilometers

      // Get a distance in miles.
      input = JOptionPane.showInputDialog("Enter " +
                          "a distance in miles.");
      miles = Double.parseDouble(input);

      // Convert the distance to kilometers.
      kilos = Metric.milesToKilometers(miles);
      JOptionPane.showMessageDialog(null,
        String.format("%,.2f miles equals %,.2f kilometers.",
                      miles, kilos));

      // Get a distance in kilometers.
      input = JOptionPane.showInputDialog("Enter " +
                       "a distance in kilometers: ");
      kilos = Double.parseDouble(input);

      // Convert the distance to kilometers.
      miles = Metric.kilometersToMiles(kilos);
      JOptionPane.showMessageDialog(null,
        String.format("%,.2f kilometers equals %,.2f miles.",
                      kilos, miles));
```

This has access to the class without creating instance of class.

5. Passing Object as argument to method

```
Rectangle box = new Rectangle (12.0, 5.0);
displayRectangle (box);
```

passing by refere

```
public static void displayRectangle(Rectangle r) {
    System.out.println("Length: " + r.getLength());
}
```

6. Returning Object from Method

```
public static void main(String[] args) {
    BankAccount account;
    account = getAccount();
    System.out.println("Account balance is " +
                       account.getBalance());
}
public static BankAccount getAccount() {
    double value = 5;
    return new BankAccount(value);
}
```

↑ return object from method

7. toString Method return a string representing the object field at any given moment

Ex

```
public class Stock {
    private String symbol;
    private double sharePrice;
    public Stock(String sym, double price){
        Symbol = sym;
        sharePrice = price;
    }
    public Strng getSymbol() {
        return symbol;
    }
    public double getSharePrice()
```

```
{   return sharePrice;
}
public String toString() {
    String str = "Trading symbol: " + symbol +
                 "\nShare price: " + sharePrice;

    return str;
}
}
```

8. This is the way to call the toString method

```
Public static void main(String[] args){
    Stock xyzCompany = new Stock("XYZ", 9.62);
    System.out.println(xzyCompany);
}
```

↑ This call the toString method automatically.

9. Equals method
   - To compare to objects the class must have a method such as equal to to compare content.

Example

```
Stock company1 = new Stock ("XyZ", 9.62);
Stock company2 = new Stocks ("XyZ", 9.62);

if ( company1 = company2)
    System.out.println("Both object are same")

else
    System.out.println("The object are different");
```

↑ This doesn't compare the object it compare only the memory address of the object

10. Example

```
public class company1 {

    public boolean equals(Stock object2 ){
      boolean status;
      if( symbol.equals(object2.symbol) && sharePrice = object2.sharePrice)
            statues=true;

       else
            statues=false;

  }
  public class StockCompare{
      public static void main(String[] args ) {
           Stock company1 = new Stock("xyz", 9.62);
           Stock company2 = new Stock("xyz", 9.62);

            if( company1.equals(company2))
                System.out.println("Both object are the same");

            else
                System.out.println("The object are different");

        }
  }
```

11. Copy constructors

- It is a constructor that accept object of the same class.

Example

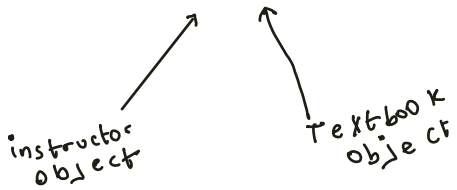
```
public class Stock {

      public Stock(Stock object2) {
```

```
        symbol = object2.symbol;
        sharePrice = object2.sharePrice;
    }
}
public static void main(String[] args){
    Stock company1 = new Stock("xyz", 9.62);
    Stock company2 = new Stock(company1);
```

12. Aggregation
    - It occurs when an instance of class is a field in another class

Example

Course class




```
public class Course {
    private
```

13. Making an instance of one class a field in another class is aggregation

14. Security issue with Aggregation
    - Do perform deep copy when creating field objects. Don't perform shallow copy

Example

```
public Course (String name, Instructor instr, TextBook text) {
        courseName = name; // shallow copy
        instructor = instr; // shallow copy
}
```

This is an example of shallow copy.

Why we don't perform shallow copy

- Other variable outside the course class might want to use the Texbook or Instructor class
- This also provide direct access to Course object's private data.

15. Return copies of field object, not the original

- When a method return a reference to field object. It should return reference to copy of field object.

Example

```
Public Instructor getInstructor() {
    return new Instructor(instructor);
}
```

↖ The right way

- There is no way it could cause code inside the class to only have access to it.

Example

```
Public Instructor getInstructor() {
    return instructor;
}
```

↖ The wrong way

16. Example of aggregation

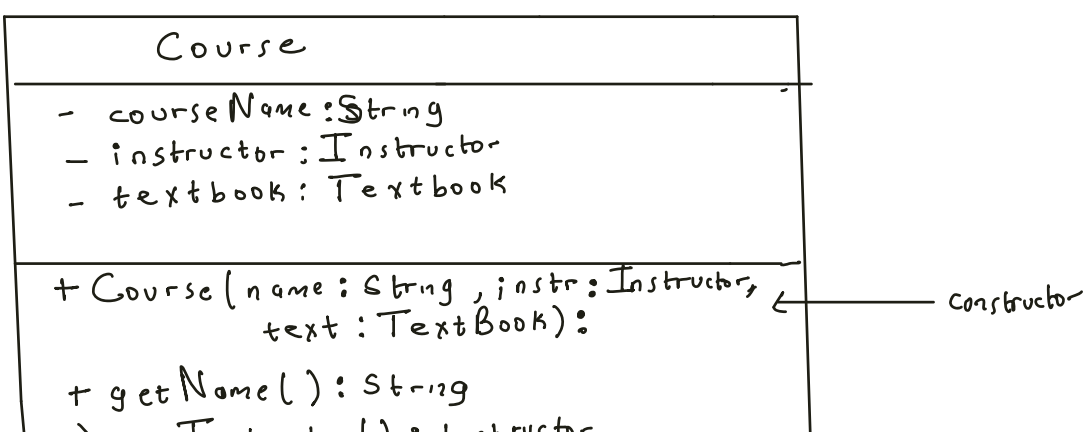
```
public class Instructor
{
   private String lastName;       // Last name
   private String firstName;      // First name
   private String officeNumber;   // Office number

   /**
      This constructor initializes the last name,
      first name, and office number.
      @param lname The instructor's last name.
      @param fname The instructor's first name.
      @param office The office number.
   */

   public Instructor(String lname, String fname,
                     String office)
   {
      lastName = lname;
      firstName = fname;
      officeNumber = office;
   }

   /**
      The copy constructor initializes the object
      as a copy of another Instructor object.
      @param object2 The object to copy.
   */

   public Instructor(Instructor object2)
   {
      lastName = object2.lastName;
      firstName = object2.firstName;
      officeNumber = object2.officeNumber;
   }

   /**
      The set method sets a value for each field.
      @param lname The instructor's last name.
      @param fname The instructor's first name.
      @param office The office number.
   */

   public void set(String lname, String fname,
                   String office)
   {
      lastName = lname;
      firstName = fname;
      officeNumber = office;
   }

   /**
      toString method
      @return A string containing the instructor
              information.
   */

   public String toString()
   {
      // Create a string representing the object.
      String str = "Last Name: " + lastName +
                   "\nFirst Name: " + firstName +
                   "\nOffice Number: " + officeNumber;

      // Return the string.
      return str;
   }
}
```

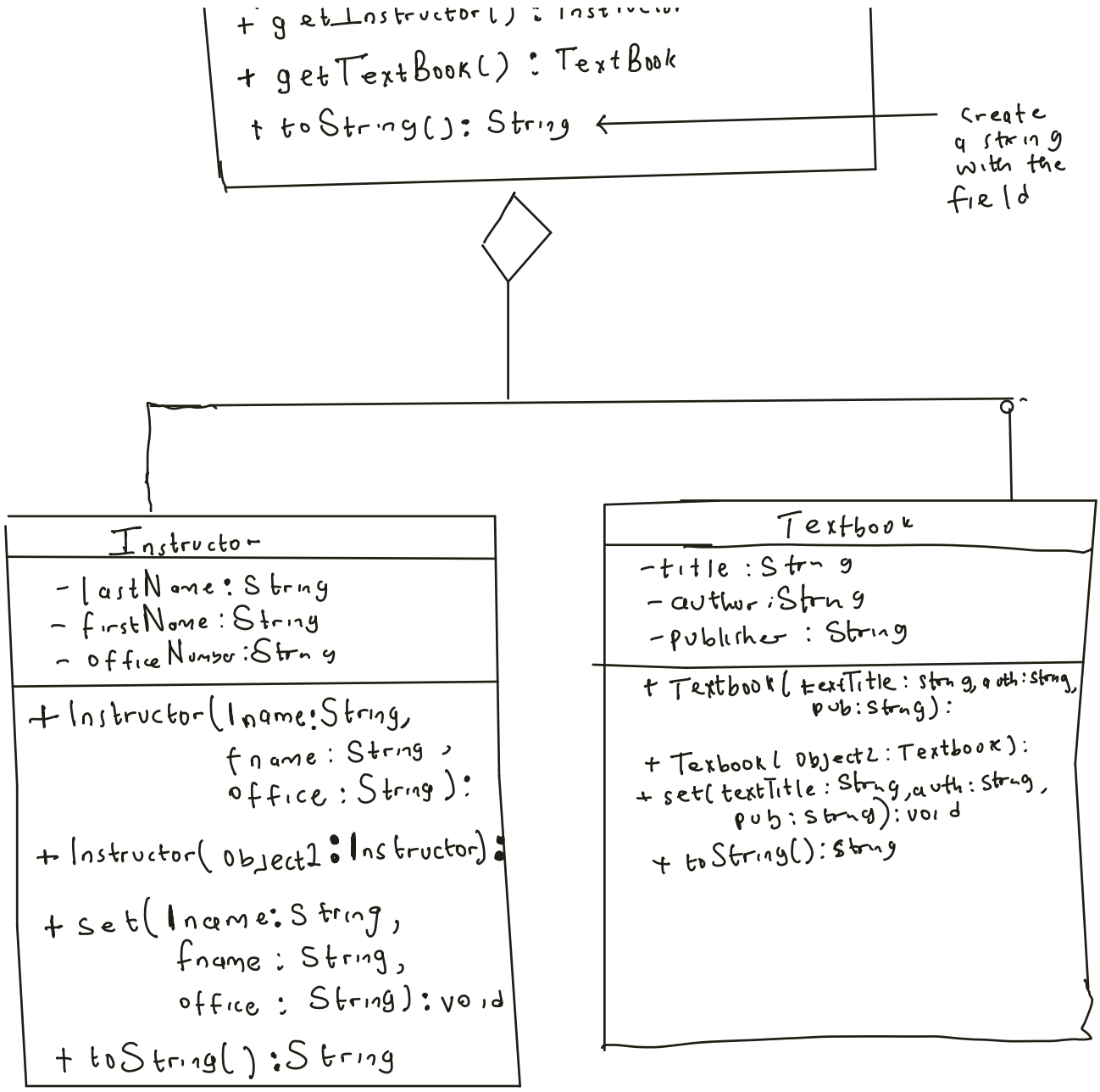
```
public class Textbook
{
   private String title;          // Title of the book
   private String author;         // Author's last name
   private String publisher;      // Name of publisher

   /**
      This constructor initializes the title,
      author, and publisher fields
      @param textTitle The book's title.
      @param auth The author's name.
      @param pub The name of the publisher.
   */

   public Textbook(String textTitle, String auth,
                   String pub)
   {
      title = textTitle;
      author = auth;
      publisher = pub;
   }

   /**
      The copy constructor initializes the object
      as a copy of another Textbook object.
      @param object2 The object to copy.
   */

   public Textbook(Textbook object2)
   {
      title = object2.title;
      author = object2.author;
      publisher = object2.publisher;
   }

   /**
      The set method sets a value for each field.
      @param textTitle The book's title.
      @param auth The author's name.
      @param pub The name of the publisher.
   */

   public void set(String textTitle, String auth,
                   String pub)
   {
      title = textTitle;
      author = auth;
      publisher = pub;
   }

   /**
      toString method
      @return A string containing the textbook
              information.
   */

   public String toString()
   {
      // Create a string representing the object.
      String str = "Title: " + title +
                   "\nAuthor: " + author +
                   "\nPublisher: " + publisher;

      // Return the string.
      return str;
   }
}
```

```
public class Course
{
   private String courseName;     // Name of the course
   private Instructor instructor;  // The instructor
   private Textbook textbook;     // The textbook

   /**
      This constructor initializes the courseName,
      instructor, and text fields.
      @param name The name of the course.
      @param instructor An Instructor object.
      @param text A Textbook object.
   */

   public Course(String name, Instructor instr,
                 Textbook text)
   {
      // Assign the courseName.
      courseName = name;

      // Create a new Instructor object, passing
      // instr as an argument to the copy constructor.
      instructor = new Instructor(instr);

      // Create a new Textbook object, passing
      // text as an argument to the copy constructor.
      textbook = new Textbook(text);
   }

   /**
      getName method
      @return The name of the course.
   */

   public String getName()
   {
      return courseName;
   }

   /**
      getInstructor method
      @return A reference to a copy of this course's
              Instructor object.
   */

   public Instructor getInstructor()
   {
      // Return a copy of the instructor object.
      return new Instructor(instructor);
   }

   /**
      getTextBook method
      @return A reference to a copy of this course's
              Textbook object.
   */

   public Textbook getTextBook()
   {
      // Return a copy of the textbook object.
      return new Textbook(textbook);
   }

   /**
      toString method
      @return A string containing the course information.
   */

   public String toString()
   {
      // Create a string representing the object.
      String str = "Course name: " + courseName +
                   "\nInstructor Information:\n" +
                   instructor +
                   "\nTextbook Information:\n" +
                   textbook;
```

```
public class CourseDemo
{
   public static void main(String[] args)
   {
      // Create an Instructor object.
      Instructor myInstructor =
          new Instructor("Kramer", "Shawn", "RH3010")

      // Create a TextBook object.
      TextBook myTextBook =
          new TextBook("Starting Out with Java",
                       "Gaddis", "Pearson");

      // Create a Course object.
      Course myCourse =
          new Course("Intro to Java", myInstructor,
                     myTextBook);

      // Display the course information.
      System.out.println(myCourse);
   }
}
```

17) Example of UML diagram





18. Avoid using null reference

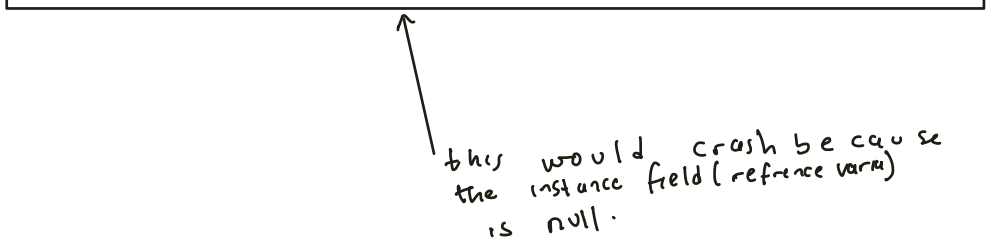
- Reference variable which is instance field is null when it doesn't reference an object.

Example

```
public class Fullname{
        Private String lastName;
        Private String firstName;

        public void setName(String las, String fir){
```

```
        lastName = las;
        firstName = fir;
    }
}
```

```
public class Demo{
    public static void main (String[] args){
        int len
        len = name.getLength();
    }
}
```



this would crash because the instance field (refrence varia) is null.

- Way to combat is is
  - Write a no-arg constructor

```
public fullname() {
    lastName = " ";
    firstName = " ";
}
```

19) Instance field - also mean reference variable

20) this variable

- it contain the address of the calling Object

instead of writing this

```
Public boolean eqval( Stock object2){
    bodean statues;
    if (symbol.equals(object2.symbol) &&
        sharePrice == object2.sharePrie)
        status = true;
    else
```

```
        status = false;
        return status;
    }
```

we could do this

```
Public boolean equal( Stock object2){
  bodean status;
  if( this.symbol.equals(object2.symbol) &&
      this.sharePrice == object2.sharePrie )
          status = true;
  else {
          status = false;
  }return status;
}
```

address of the calling object

21. Using this to prevent over shadowing

- sometime we want to give parameter and class field the same name to overcome shadowing we use this.

Example

```
Public Stock {
   private String symbol;
   private double price;

   public Stock(String symbol, double price){
        this.symbol = symbol;
        this.price = price;
   }
```

this calls the parameter variable

this called the field symbol

22. calling a constructor from another constructor

```
class Stock {
   Public Stock (String sym, double price){
        symbol = sym;
        sharePriee = price;
   }

   public Stock (String sym) {
        this(sym, 0.0);
```

calls this constructor because it has

two parameter passed in

}

23. enumerated data type is a form of data type we created with our own form of rules

Example

```
enum Day { SUNDAY , MONDAY }
```

- Day: form of class (Data type)
- SUNDAY, MONDAY: (CONSTANT) type of data it could store
- Try to write in all uppercase

24. Declaring a variable of enumerated type

```
Day workDay;
workDay = Day.Wednesday;
```

- Day: Datatype
- workDay: variable
- = Day.Wednesday: assigning a value (this assign the address)

25. enumerated type
- When enumerated type declaration is done we are creating a special type of class
- The enum constant listed inside the brace are the object of the class

26. `enum Day { SUNDAY, MONDAY }`
- Day: form of class (data type)

27. workDay = Day.WEDNESDAY
- it assign it address of the object to the variable

Example

object of Day type

Day.SUNDAY

The workDay variable hold Day.MONDAY

28. enum constant are equipped with few method
    - One is the toString method
        - It return name of the calling enum constant as string

Example

```
Day workDay = Day.WEDNESDAY
                  ↑ calling

System.out.println(workDay)
                   ↑ return name of calling enum
```

output

WEDNESDAY

29. Ordinal method return value representing the constant ordinal value (index).

```
enum Day { SUNDAY(0), MONDAY(1), TUESDAY(2) }

Day workDay = Day.TUESDAY
System.out.println(workDay.ordinal());
```

output

2

30. Another enum data type method is equal
    - it is used to compare to see if object is equal to enum calling constant

Example

```
Day myDay = Day.Tuesday
if (myDay.equals(Day.Tuesday))
    System.out.println("The same");
```

31) Another is the data type method to Compare
- if calling costant' ordinal value > than argument ordinal value

It is positive

Example

```
Day myDay = Day.FRIDAY;
if (myDay.compareTo(Day.TUESDAY) > 0){
    System.out.println(myDay + " it is greather" + "than" + Day.TUESDAY);
```

32. We can write an enumerated type in it own class

Example

(CarType.java)

```
enum CarType{PORSCHE, FERRARI, JAGUAR};
enum CarColor{RED, BLACK, BLUE, SILVER};
```

enum class

(SportsCar.java)

```
class SportsCar {
    private CarType make;
    private CarColor color;
    private double price;

    public SportsCar(CarType aMake, CarColor aColor, double aPrice){
        make = aMake;
        color = aColor;
        price = aPrice;
    }
    public CarType getCarType(){
        return make;
    }
    public CarColor getCarColor(){
        return color;
    }
    public double getPrice(){
```

using it in a class

```
public double ge...
        return price;
    }
    public String toString() {
        String word = "The car type is " + make +
                      "the car color is " + color +
                      "the car price is " + price;

        return word;
    }
}
```

(Demo.java)

```
public class Demo {
    public static void main(String[] args) {
        SportsCar yourNewCar = new SportsCar(CarType.PORSCHE,
                                             CarColor.RED,
                                             1000);
    }
}
```

passing the enumerated value

33. Garage Collection

- It remove un reference object from memory

Example

```
BankAccount account1 = new BankAccount(5);
BankAccount account2 = account1;
account1 = null;
```

↖ it doesn't reference any object again. Garge collector could be called any time. account2 still reference a object

```
account2 = null;
```

↖ it doesn't reference any object again

Chapter 9 Text Processing and More Wrapper Class

Text Processing and more about wrapper class

1. Wrapper class is a primative data type that allow you to create object instead of variable
2. The wrapper class are provided with method that perform operation
3. Wrapper class are immutable ones created they can't change
4. One example of the wrapper class is the character class.

5. Character class
   - it is a wrapper class for char data type

6. Some Character class method
   a. Boolean isDigit(Char ch) ← return true if it digit (what we test to see with)
   b. Boolean isWhitespace((char ch); ← return true if it white space

7.) public class Character{

```
public static void main(String[] args){
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Enter any character");
    String input = keyboard.nextLine();
    Char ch = input.charAt(0);
    if (Character.isLetter(ch){
        System.out.println("This is a letter");
    }
```

This remove just one char at index 0

Output:

display — Enter any character
input — Name
output — This is a letter

8. Character case conversion
   a) Character.toLowerCase(ch); ← The character

b) Character . toUpperCase (ch);

9. The character could be use in a do-while loop

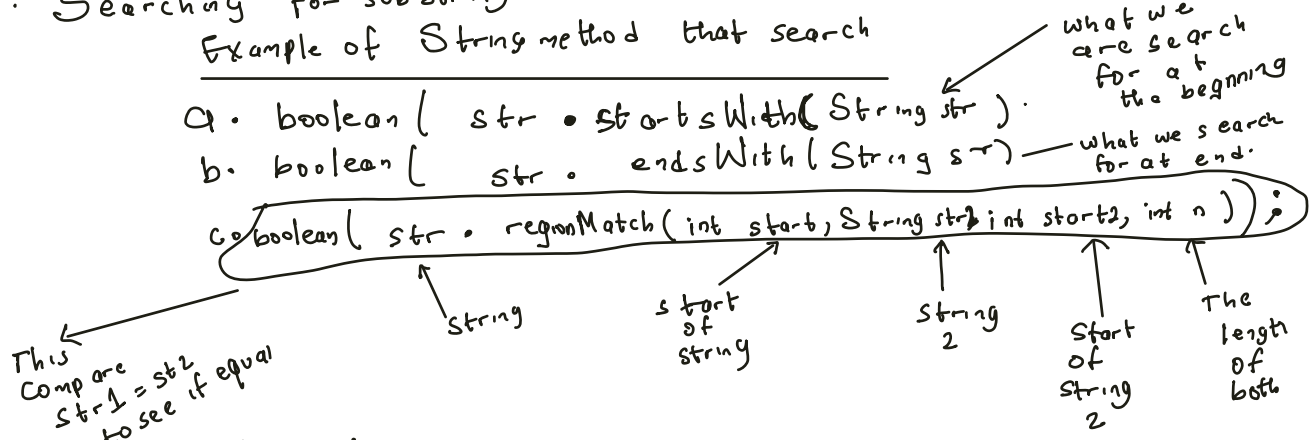
```
do {
    System.out.println("Enter yes to continue " +
                       "No to not continue ");
    String input = keyboard.nextLine();
    Char ch = input.charAt(0);
} while ( Character.toUpperCase(ch) == 'Y ');
```

10. String methods.
    - Many method for with with or searching String class

11. Searching for subString

Example of String method that search

a. boolean (str . startsWith(String str) .
b. boolean (str . endsWith (String str)
c. boolean (str . regionMatch (int start, String str2, int start2, int n));



Example

```
String str = "four width ";
if( str.startsWith("four") ){
    System.out.println("The string start with four ");
```

Example

```
String[] people = { "Custew Hill", "Davis",
```

```
"Jones", "Davis Lee"};

System.out.println("Enter what you are searching");
String lookUp = keyboard.nextLine();
System.out.println("This are the match");
for (String person: people){
    if (person.startsWidth(lookUp)){
        System.out.println(person);
    }
}
```

Output

output ——— Enter what you searching
input ——— Davis
output ——— Here are the matches
Davis
Davis Lee

12. Example

```
String str = "four score and seven years ago";
String str2 = "Those seven years passed quickly";

if (str.regionMatches(15, str2, 6, 11))
    System.out.println("The region match");

else
    System.out.println("The region doesn't match");
```

13. String method to get character or substring location
- int indexOf(Char ch) ← the index of the first ch
- int indexOf(char ch, int start) ← start position to start searching for character

- int indexOf (String str)
  ↑ the starting index of where the character was found
- int indexOf (String str, int start)
  ↑ Start position to search for the starting index of string.
- int lastIndexOf (Char ch)
  ↑ Search for string object for the character for in last occurence position(index)

14. Example

```
String str = "four score and seven years ago";
first = str.indexOf ('r');
last = str.lastOf ('r');
System.out.println ("The first index where it was found is "+
                    first );
```

15. The use of while loop to show each position of 'r'

```
String str = "Four score and seven years ago";
int position;
System.out.println("The letter r appear at the " + "following" +
                   " locations: ");

position = str.indexOf ('r');
 while (position != -1){
    System.out.println (position);
    position = str.indexOf('r', position+1);

 }
```

Output

3
8
24

16. String method to extract substrings

- String substring(int start) ← get string from starting position to the end
- String substring(int start, int end)
  get string from starting position to stated end
- void getChar(int start, int end, char[] array, int arrayStart)
- char[] toCharArray()

17. Example

```
String name = "Cynthia Susan Lee";
String firstName = name.substring(14);
String lastName = name.substring(0, 6);
System.out.println("My firstname is " + firstName +
                   "My lastname is " + lastName);
```

Output

Lee Cynthia

18. Example

```
String fullName = "Cynthia Susan Lee";
char[] nameArray = new char[5];
fullName.getChars(7, 13, nameArray, 0);
System.out.println("The value in the array are: ");
for (int i=0; i < nameArray.length; i++)
    System.out.println(nameArray[i] + " ");
```

output

The value of the array are

Susan

19. Example

```
String fullName = "Cynthia Susan Lee";
char[] nameArray;
nameArray = fullName.toCharArray();
System.out.println("the value in the array are: ");
for(int i = 0; i < nameArray.length; i++)
    System.out.println(nameArray[i] + " ");
```

output

C y n t h i a S u s a n L e e

20. Method that return a modified String

- String concat(String str) ← content of str concatenated to string object
- String replace(char oldChar, char newChar)
- String trim() ← remove any leading or trailing whitespace.

Example

```
String fullName;
String firstName = "Lee";
String lastName = "Sarah";
fullName = lastName.concat(firstName);
System.out.print(fullName);
```

output

Sarah Lee

21. valueOf method

- it accept a value of any primative data type and return a string representation of value

Example

String valueOf(long number) — from long to String

String valueOf (int number) ___ from int to String

String valueOf (char[] array) ___ from char array to String

String valueOf (char[] array, int subscript, int count) ___ from char array to string

- subscript: starting point
- count: number of element

Example

```
boolean b = true;
char[] letter = {a, b, c};
double d = 2.49
int i = 7;
System.out.println(String.valueOf(b));
System.out.println(String.valueOf(letter));
System.out.println(String.valueOf(letter, 1, 3));
```

output

```
true
abc
abc
```

22. String is similar to StringBuilder, except StringBuilder let you change the content.

23. String are immutable you can't change it content.

Example

```
String name = "Sarah";
name = "Jane";
```

we can do this because we are assign it to different address

24. StringBuilder constructor

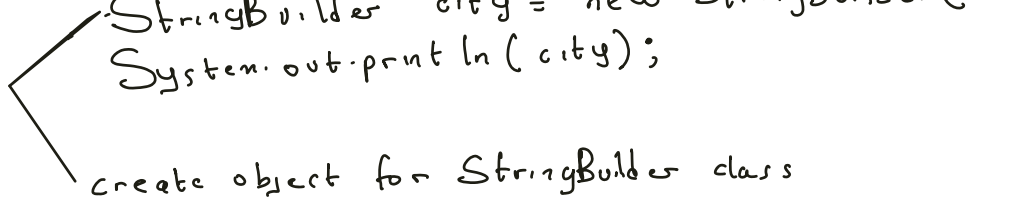
a) StringBuilder() ← object enough

25. String builder Constructor

StringBuilder( ) – This accept no argument • it give object space to hold 16 character

StringBuilder(int length) – This accept argument on amount of space to hold characters

StringBuilder(String str) – This accept string as argument + the intial 16 characters space



26) Example

```
StringBuilder city = new StringBuilder("Charleston");
System.out.println(city);
```

create object for StringBuilder class

27. Method of same that StringBuilder has like String

```
char charAt(int position)
void getChars(int start, int end, char[] array, int arrayStart)
int indexOf(String str)
int indexOf(String str, int start)
int lastIndexOf(String str, int start)
int length()
String substring(int star)
String substring(int start, int end)
```

26. Append method for StringBuilder class

Example

```
StringBuilder str = new StringBuilder();
str.append("We are ");
str.append("are");
System.out.println(str);
```

output

We are are

27. insert method for StringBuilder class
- insert at a specific point

Example

```
StringBuilder str = new StringBuilder("New City");
str.insert(4, "York");
System.out.println(str);
```

output

New York City

28 replace method
- this replace a specific place in the StringBuilder class

Example

```
StringBuilder str = new StringBuilder("We moved");
str.replace(3, 7, nowss);
System.out.println(str);
```

output

We nowss

29 delete method
- this delete a substring from StringBuilder object
- StringBuilder delete(int start, int end);
- StringBuilder deleteCharAt(int position);

Example

```
StringBuilder str = new StringBuilder("I ate");
System.out.println(str);
str.deleteCharAt(0);
System.out.println(str);
```

output

I ate
ate

29. Tokenizing Strings

- It is the breaking down of string into componenent

30. Example of Tokenizing string

"peach raspberry strawberry vanilla"

- The character that seperate it is the space

31. How to tokenizing string

Example

```
Str str = "One two three four";
String[] token = str.split(" ");
for(String s: token) {
    System.out.println(s);
}
```

output

```
One
two
three
four
```

32. How multi-character delimeter(split)

Ex

```
String str = joe@gaddisbook.com
String[] token = str.split("[@.]");
for(String s :token) {
    System.out.println(s);
}
```

output

```
Joe
gaddisbook
com
```

33. Numeric data type are wrapper class.
- it is provided with method that perform useful operation

34. toString Methods
- convert a int to a string

Example
```
int i = 12
String str1 = Integer.toString(i); ← convert to string
System.out.println(str1);
```
output
```
12
```

35 MIN_VALUE and MAX_VALUE constants
- it hold the min value or max value a numeric data type can hold

Example
```
System.out.println("The minimum value for an " + "int is " +
                   Integer.MIN_VALUE);
```

output
```
-2147483648
```

36. Creating object for wrapper class (Autoboxing)
```
Integer number = new Integer(7);
```
↑ create an integer object and initilize it to 7

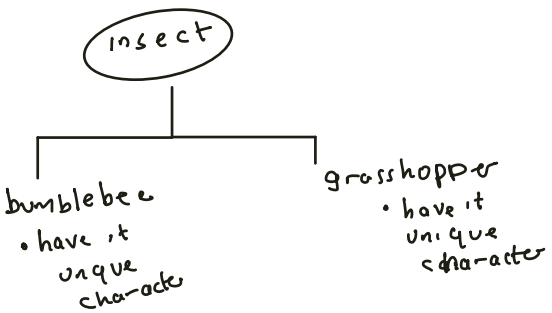
39 Storing wrapper class object in ArrayList
```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(myInt);
```

Chapter 10 Inheritance

1. Inheritance allow a new class to extend an existing class

2. Inheritance also mean new class inherit the member of class it extend

3 Example
A car is a vehicle -
A flower is a plant



4. When is a relationship exist it mean specialized object has all of the characteristic of the general object

5. inheritance involve a super-class and subclass

6. Superclass is the general class and subclass is extended version of superclass

7. Superclass can also be called base class
Subclass can be called derived class

8. Example
class - GradeActivity.java

```
public class GradedActivity {
        private double score;
        public void setScore(double s) {
            score = s;
        }
        public char getGrade() {
            char letterGrade
            if( score >= 90 )
```

SuperClass
• Because it has all the characteristic

```
        letterGrade = 'A';
    else
        letterGrade = 'F';
    return letterGrade;
  }
}
```

inheritance of (GradedActivity)

class : FinalExam.java

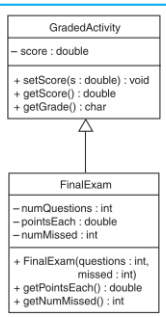
```
public class FinalExam extends GradedActivity {
  private int numQuestion;
  private double pointsEach;
  private int numMissed;

  public FinalExam(int questions, int missed) {
      double numericScore;
      numQuestion = questions;
      numMissed = missed;
      pointsEach = 100.0 / questions;
      numericScore = 100 - (missed * pointsEach);
      setScore(numericScore);
  }
  public double getPointsEach() {
      return pointsEach;
  }
  public int getNumMissed() {
      return numMissed;
  }
}
```

↑ Subclass
- it inherit all of the GradedActivity class, all of the public members of GradedActivity class

```
import java.util.Scanner;
public class FinalExamDemo {
    public static void main(String[] args) {
```

```
Scanner Keyboard = new Scanner(System.in);

String input;
int question;
int missed;
System.out.println("Enter the nomber of question");
question = Keyboard.nextInt();
System.out.println("Enter the question missed");
missed = Keyboard.nextInt();
FinalExam exam = new FinalExam(question, missed);
System.out.println(null, "Each question count" +
                   exam.getPointsEach() +
                   "points.");
System.out.println("The exam score is " +
                   exam.getScore());
System.out.println("The exam grade is " +
                   exam.getGrade());
}
}
```

9. inheritance in UML Diagram



10. In inheritance the super-class constructor excute before the subclasse constructor

Example

Class - SuperClass.java

```
public class SuperClass {
```

```
    public SuperClass() {
        System.out.println("This is the " + "super-class constructor");
    }
}
```

Class - SuberClas.java

```
public class SubClass extends SuperClass {
    public SubClass(){
        System.out.println("This is the " + "subclass constructor");
    }
}
```

Main class - ConstructorDemo.java

```
public class ConstructorDemo {
    public static void main(String[] args) {
        SubClass obj = new SubClass();
    }
}
```

output

```
This is the superClass constructor
This is the subclass constructor
```

11. Calling the superclass Constructor
- The super keyword class the superclass constructor
- The super can key word can be used to access member of the super-Class

12. Example
class - SuperClass.java

```
public class SuperClass {
```

```
    public SuperClass() {
        System.out.println("no arg constructor");
    }
    public SuperClass(int arg) {
        System.out.println("value pass is " + arg);
    }
}
```

class - SubClass.java

```
public class SubClass extends SuperClass {
    public SubClass() {
        super(10);
        System.out.println("This is the the subclass" +
                           "constructor");
    }
}
```

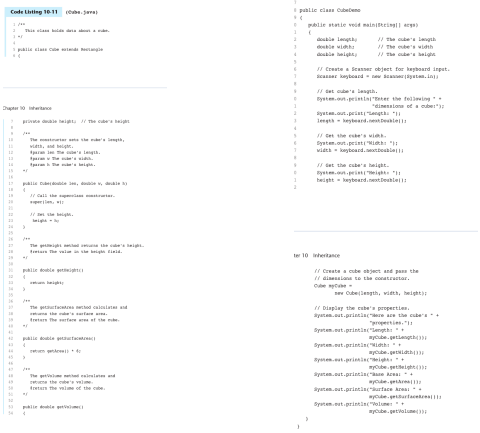
class - ConstructorDemo.java

```
public class ConstructorDemo {
    public static void main(String[] args) {
        Subclass obj = new SubClass();
    }
}
```

output

Value passed is 10
This is the subclass constructor

13. Example of SuperClass constructor and SubClass constructor



Code Listing 10-11 (Cube.java)

```
/**
   This class holds data about a cube.
*/

public class Cube extends Rectangle
{
```

Chapter 10 Inheritance

```
   private double height;  // The cube's height

   /**
      The constructor sets the cube's length,
      width, and height.
      @param s The cube's length.
      @param w The cube's width.
      @param h The cube's height.
   */

   public Cube(double s, double w, double h)
   {
      // Call the superclass constructor.
      super(s, w);

      // Set the height.
      height = h;
   }

   /**
      The getHeight method returns the cube's height.
      @return The value in the height field.
   */

   public double getHeight()
   {
      return height;
   }

   /**
      The getSurfaceArea method calculates and
      returns the cube's surface area.
      @return The surface area of the cube.
   */

   public double getSurfaceArea()
   {
      return getArea() * 6;
   }

   /**
      The getVolume method calculates and
      returns the cube's volume.
      @return The volume of the cube.
   */

   public double getVolume()
   {
```

```
}

public class CubeDemo
{
   public static void main(String[] args)
   {
      double length;      // The cube's length
      double width;       // The cube's width
      double height;      // The cube's height

      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Get cube's length.
      System.out.println("Enter the following " +
                         "dimensions of a cube:");
      System.out.print("Length: ");
      length = keyboard.nextDouble();

      // Get the cube's width.
      System.out.print("Width: ");
      width = keyboard.nextDouble();

      // Get the cube's height.
      System.out.print("Height: ");
      height = keyboard.nextDouble();
```

ter 10 Inheritance

```
      // Create a cube object and pass the
      // dimensions to the constructor.
      Cube myCube =
            new Cube(length, width, height);

      // Display the cube's properties.
      System.out.println("Here are the cube's " +
                         "properties:");
      System.out.println("Length: " +
                         myCube.getLength());
      System.out.println("Width: " +
                         myCube.getWidth());
      System.out.println("Height: " +
                         myCube.getHeight());
      System.out.println("Base Area: " +
                         myCube.getArea());
      System.out.println("Surface Area: " +
                         myCube.getSurfaceArea());
      System.out.println("Volume: " +
                         myCube.getVolume());
   }
}
```

(4) Overidding SuperClass Method

- A subclass may have a method with same signature as SuperClass method. In such case a subclass override a superclass method

Example

Class – GradedActivity.java

```
public class GradedActivity {
      private double score;

      public void setScore(double s){
            score = s;
      }

      public void getScore( ){
            return score;
      }

      public char getGrade( ) {
            char letterGrade;
            if( score >= 90)
                  letterGrade = 'A';
            else( score >= 80)
                  letterGrade = 'B';

            return letterGrade;
      }
}
```

class - CurvedActivity.java

```
public class CurvedActivity extends GradeActivity
{
      double rawScore;
      double percentage;

      public CurvedActivity (double percent)
      {
            percentage = percent;
            rawScore = 0.0;
      }

      @Override
      public void setScore(double s){
            rawScore = s;
            super.setScore(rawScore *percentage);
      }
      public double getRawScore(){
            return rawScore;
      }
      public double getPercentage(){
            return percentage;
      }
}
```

This override the superclass to see this when the function is called

```
1  import java.util.Scanner;
2
3  /**
4     This program demonstrates the CurvedActivity class,
5     which inherits from the GradedActivity class.
6  */
7
8  public class CurvedActivityDemo
9  {
0     public static void main(String[] args)
1     {
2        double score;              // Raw score
3        double curvePercent;       // Curve percentage
4
5        // Create a Scanner object to read keyboard input.
6        Scanner keyboard = new Scanner(System.in);
7
8        // Get the unadjusted exam score.
9        System.out.print("Enter the student's " +
0                         "raw numeric score: ");
1        score = keyboard.nextDouble();
2
3        // Get the curve percentage.
4        System.out.print("Enter the curve percentage: ");
5        curvePercent = keyboard.nextDouble();
6
7        // Create a CurvedActivity object.
8        CurvedActivity curvedExam =
9                    new CurvedActivity(curvePercent);
```

main class that excute it

Program Output with Example Input Shown in Bold

```
Enter the student's raw numeric score: 87 [Enter]
Enter the curve percentage: 1.06 [Enter]
The raw score is 87.0 points.
The curved score is 92.22
The exam grade is A
```

output

10.3 Overridi

```
30
31       // Set the exam score.
32       curvedExam.setScore(score);
33
34       // Display the raw score.
35       System.out.println("The raw score is " +
36                          curvedExam.getRawScore() +
37                          " points.");
38
39       // Display the curved score.
40       System.out.println("The curved score is " +
41                          curvedExam.getScore());
42
43       // Display the exam grade.
44       System.out.println("The exam grade is " +
45                          curvedExam.getGrade());
46    }
47 }
```

15. Protected member can be access by method in same class or method of a subclass.

class - Score

```
public class Score {
    protected int score;

    public int getScore() {
        return score;
    }
}
```

class - Exam

```
public class Exam Extends Score {

    public void setScores(int scores){
        score = scores;
    }
}
```

Main Class

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

```
Exam ex = new Exam( );
ex.setScores(2);
System.out.println(ex.getScore());
}
}
```

16 Example

```
4
5 public class GradedActivity2
6 {
7    protected double score;  // Numeric score
8
9    /**
10      The setScore method sets the score field.
11      @param s The value to store in score.
12   */
13
14   public void setScore(double s)
15   {
16      score = s;
17   }
18
19   /**
20      The getScore method returns the score.
21      @return The value stored in the score field.
22   */
23
24   public double getScore()
25   {
26      return score;
```

apter 10 Inheritance

```
27   }
28
29   /**
30      The getGrade method returns a letter grade
31      determined from the score field.
32      @return The letter grade.
33   */
34
35   public char getGrade()
36   {
37      char letterGrade;
38
39      if (score >= 90)
40         letterGrade = 'A';
41      else if (score >= 80)
42         letterGrade = 'B';
43      else if (score >= 70)
44         letterGrade = 'C';
45      else if (score >= 60)
46         letterGrade = 'D';
47      else
48         letterGrade = 'F';
49
50      return letterGrade;
51   }
52 }
```

```
5
6 public class FinalExam2 extends GradedActivity2
7 {
8    private int numQuestions;    // Number of questions
```

10.4 Protected

```
10   private double pointsEach;   // Points for each question
11   private int numMissed;       // Number of questions missed
12
13   /**
14      The constructor sets the number of questions on the
15      exam and the number of questions missed.
16      @param questions The number of questions.
17      @param missed The number of questions missed.
18   */
19
20   public FinalExam2(int questions, int missed)
21   {
22      double numericScore;       // To hold a numeric score
23
24      // Set the numQuestions and numMissed fields.
25      numQuestions = questions;
26      numMissed = missed;
27
28      // Calculate the points for each question and
29      // the numeric score for this exam.
30      pointsEach = 100.0 / questions;
31      numericScore = 100.0 - (missed * pointsEach);
32
33      // Call the inherited setScore method to
34      // set the numeric score.
35      setScore(numericScore);
36
37      // Adjust the score.
38      adjustScore();
39   }
40
41   /**
42      The getPointsEach method returns the number of
43      points each question is worth.
44      @return The value in the pointsEach field.
45   */
46
47   public double getPointsEach()
48   {
49      return pointsEach;
50   }
51
52   /**
53      The getNumMissed method returns the number of
54      questions missed.
55      @return The value in the numMissed field.
56   */
57
```

hapter 10 Inheritance

```
58   public int getNumMissed()
59   {
60      return numMissed;
61   }
62
63   /**
64      The adjustScore method adjusts a numeric score.
65      If score is within 0.5 points of the next whole
66      number, it rounds the score up.
67   */
68
69   private void adjustScore()
70   {
71      double fraction;
72
73      // Get the fractional part of the score.
74      fraction = score - (int) score;
75
76      // If the fractional part is .5 or greater,
77      // round the score up to the next whole number.
78      if (fraction >= 0.5)
79         score = score + (1.0 - fraction);
80   }
81 }
```

```
7
8 public class ProtectedDemo
9 {
10   public static void main(String[] args)
11   {
12      String input;         // To hold input
13      int questions;        // Number of questions
14      int missed;           // Number of questions missed
15
```

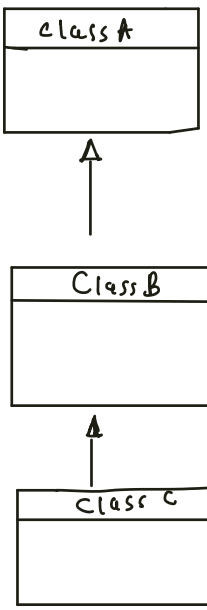
10.4 Protected M

```
16      // Get the number of questions on the exam.
17      input = JOptionPane.showInputDialog("How many " +
18                  "questions are on the final exam?");
19      questions = Integer.parseInt(input);
20
21      // Get the number of questions the student missed.
22      input = JOptionPane.showInputDialog("How many " +
23                  "questions did the student miss?");
24      missed = Integer.parseInt(input);
25
26      // Create a FinalExam object.
27      FinalExam2 exam = new FinalExam2(questions, missed);
28
29      // Display the test results.
30      JOptionPane.showMessageDialog(null,
31            "Each question counts " + exam.getPointsEach() +
32            " points.\nThe exam score is " +
33            exam.getScore() + "\nThe exam grade is " +
34            exam.getGrade());
35
36      System.exit(0);
37   }
38 }
```

← main Class

17. Chain of Inheritance
- A super class can also be inherited from another class

Example



- Class B inherited class A
- Class C inherited class B

18. Example of chain inheritance

(1)

```
public class GradedActivity2
{
   protected double score;  // Numeric score

   /**
      The setScore method sets the score field.
      @param s The value to store in score.
   */

   public void setScore(double s)
   {
      score = s;
   }

   /**
      The getScore method returns the score.
      @return The value stored in the score field.
   */

   public double getScore()
   {
      return score;
```

apter 10 Inheritance

```
   }

   /**
      The getGrade method returns a letter grade
      determined from the score field.
      @return The letter grade.
   */

   public char getGrade()
   {
      char letterGrade;

      if (score >= 90)
         letterGrade = 'A';
      else if (score >= 80)
         letterGrade = 'B';
      else if (score >= 70)
         letterGrade = 'C';
      else if (score >= 60)
         letterGrade = 'D';
      else
         letterGrade = 'F';

      return letterGrade;
   }
}
```

(2)

```
public class PassFailActivity extends GradedActivity
{
   private double minPassingScore; // Minimum passing score

   /**
      The constructor sets the minimum passing score.
      @param mps The minimum passing score.
   */

   public PassFailActivity(double mps)
   {
      minPassingScore = mps;
   }

   /**
      The getGrade method returns a letter grade
      determined from the score field. This
      method overrides the superclass method.
      @return The letter grade.
   */

   @Override
   public char getGrade()
   {
```

10.5 I

```
      char letterGrade;

      if (super.getScore() >= minPassingScore)
         letterGrade = 'P';
      else
         letterGrade = 'F';

      return letterGrade;
   }
}
```

(3)

```
public class PassFailExam extends PassFailActivity
{
   private int numQuestions;      // Number of questions
   private double pointsEach;     // Points for each question
   private int numMissed;         // Number of questions missed

   /**
      The constructor sets the number of questions, the
      number of questions missed, and the minimum passing
      score.
      @param questions The number of questions.
      @param missed The number of questions missed.
      @param minPassing The minimum passing score.
   */

   public PassFailExam(int questions, int missed,
                       double minPassing)
```

ter 10 Inheritance

```
   {
      // Call the superclass constructor.
      super(minPassing);

      // Declare a local variable for the score.
      double numericScore;

      // Set the numQuestions and numMissed fields.
      numQuestions = questions;
      numMissed = missed;

      // Calculate the points for each question and
      // the numeric score for this exam.
      pointsEach = 100.0 / questions;
      numericScore = 100.0 - (missed * pointsEach);

      // Call the superclass's setScore method to
      // set the numeric score.
      setScore(numericScore);
   }

   /**
      The getPointsEach method returns the number of
      points each question is worth.
      @return The value in the pointsEach field.
   */

   public double getPointsEach()
   {
      return pointsEach;
   }

   /**
      The getNumMissed method returns the number of
      questions missed.
      @return The value in the numMissed field.
   */

   public int getNumMissed()
   {
      return numMissed;
   }
}
```

(4)

```
public class PassFailExamDemo
{
   public static void main(String[] args)
   {
      int questions;       // Number of questions
      int missed;          // Number of questions missed
      double minPassing;   // Minimum passing score

      // Create a Scanner object for keyboard input.
      Scanner keyboard = new Scanner(System.in);

      // Get the number of questions on the exam.
      System.out.print("How many questions are " +
                       "on the exam? ");
      questions = keyboard.nextInt();

      // Get the number of questions missed.
      System.out.print("How many questions did " +
                       "the student miss? ");
      missed = keyboard.nextInt();

      // Get the minimum passing score.
      System.out.print("What is the minimum " +
                       "passing score? ");
      minPassing = keyboard.nextDouble();

      // Create a PassFailExam object.
      PassFailExam exam =
          new PassFailExam(questions, missed, minPassing);

      // Display the points for each question.
      System.out.println("Each question counts " +
                         exam.getPointsEach() + " points.");

      // Display the exam score.
      System.out.println("The exam score is " +
                         exam.getScore());

      // Display the exam grade.
```

19. Object Class

- When a class doesn't use extends key word to inherit from another class Java automatically extend it from the object class

Example

```
public class MyClass extends object {

}

public class MyClass {

}
```

if the some

20. Every class inherit the object class member.
    - toString method — return @ followed by object hash code
    - equal method — accept a reference to an object. And return true if the argument reference the calling object

21) Example

```
public class PassFailExam
{
   private int numQuestions;      // Number of questions
   private double pointsEach;     // Points for each question
   private int numMissed;         // Number of questions missed

   /**
      The constructor sets the number of questions, the
      number of questions missed, and the minimum passing
      score.
      @param questions The number of questions.
      @param missed The number of questions missed.
      @param minPassing The minimum passing score.
   */

   public PassFailExam(int questions, int missed,
                       double minPassing)
```

```
Ex# 10  Inheritance

   {
      // Call the superclass constructor.
      super(minPassing);

      // Declare a local variable for the score.
      double numericScore;

      // Set the numQuestions and numMissed fields.
      numQuestions = questions;
      numMissed = missed;

      // Calculate the points for each question and
      // the numeric score for this exam.
      pointsEach = 100.0 / questions;
      numericScore = 100.0 - (missed * pointsEach);

      // Call the superclass's setScore method to
      // set the numeric score.
      setScore(numericScore);
   }

   /**
      The getPointsEach method returns the number of
      points each question is worth.
      @return The value in the pointsEach field.
   */

   public double getPointsEach()
   {
      return pointsEach;
   }

   /**
      The getNumMissed method returns the number of
      questions missed.
      @return The value in the numMissed field.
   */

   public int getNumMissed()
   {
      return numMissed;
   }
}
```

```
public class ObjectMethods
{
   public static void main(String[] args)
   {
      // Create two objects.
      PassFailExam exam1 =
```

```
                  new PassFailExam(0, 0, 0);
      PassFailExam exam2 =
                  new PassFailExam(0, 0, 0);

      // Send the objects to println, which
      // will call the toString method.
      System.out.println(exam1);
      System.out.println(exam2);

      // Test the equals method.
      if (exam1.equals(exam2))
         System.out.println("They are the same.");
      else
         System.out.println("They are not the same.");
   }
}
```

Output

PassFailExam@15f0472

PassFailExam@18d107f
They are not the same.

22. Changes the behavior of this method you must override it in the class

23. Polymorphism
- It is the ability to reference object different from it own as long as those type are subclass of its type

Example

```
GradedActivity exam1 = new FinalExam(50, 7);
GradedActivity exam2 = new PassFailActivity(70);
```

- This are legal because because FinalExam, PassFailActivity class inherit from GradedActivity.

- The GradedActivity variable can call only the three method of the object variable reference. They are the setScore, getScore, and getGrade.

Example

```
GradedActivity exam1 = new FinalExam(100, 10, 70);
System.out.println(exam.getScore()); // This work
System.out.println(exam.getGrade()); This work.
System.out.println(exam.getPointsEach()); // ERROR won't work
```

↑ why is because exam variable know only method in GradedActivity class.

24. Example

```
GradedActivity exam = new PassFailActivity(60);
exam.setScore(70);
System.out.println(exam.getGrade());
```

- The passFailActivity class extended the GradedActivity and

overide the getGrade method. When the last excute it calles the PassfailActivity getGrade version

25. Example of Polymorphism

```
4
5  public class GradedActivity2
6  {
7     protected double score;  // Numeric score
8
9     /**
10       The setScore method sets the score field.
11       @param a The value to store in score.
12    */
13
14    public void setScore(double s)
15    {
16       score = s;
17    }
18
19    /**
20       The getScore method returns the score.
21       @return The value stored in the score field.
22    */
23
24    public double getScore()
25    {
26       return score;
```

```
apter 10  Inheritance

27    }
28
29    /**
30       The getGrade method returns a letter grade
31       determined from the score field.
32       @return The letter grade.
33    */
34
35    public char getGrade()
36    {
37       char letterGrade;
38
39       if (score >= 90)
40          letterGrade = 'A';
41       else if (score >= 80)
42          letterGrade = 'B';
43       else if (score >= 70)
44          letterGrade = 'C';
45       else if (score >= 60)
46          letterGrade = 'D';
47       else
48          letterGrade = 'F';
49
50       return letterGrade;
51    }
52 }
```

```
5
6  public class PassFailExam extends PassFailActivity
7  {
8     private int numQuestions;     // Number of questions
9     private double pointsEach;    // Points for each question
10    private int numMissed;        // Number of questions missed
11
12    /**
13       The constructor sets the number of questions, the
14       number of questions missed, and the minimum passing
15       score.
16       @param questions The number of questions.
17       @param missed The number of questions missed.
18       @param minPassing The minimum passing score.
19    */
20
21    public PassFailExam(int questions, int missed,
22                        double minPassing)
```

```
er 10  Inheritance

1     {
2        // Call the superclass constructor.
3        super(minPassing);
4
5        // Declare a local variable for the score.
6        double numericScore;
7
8        // Set the numQuestions and numMissed fields.
9        numQuestions = questions;
10       numMissed = missed;
11
12       // Calculate the points for each question and
13       // the numeric score for this exam.
14       pointsEach = 100.0 / questions;
15       numericScore = 100.0 - (missed * pointsEach);
16
17       // Call the superclass's setScore method to
18       // set the numeric score.
19       setScore(numericScore);
20    }
21
22    /**
23       The getPointsEach method returns the number of
24       points each question is worth.
25       @return The value in the pointsEach field.
26    */
27
28    public double getPointsEach()
29    {
30       return pointsEach;
31    }
32
33    /**
34       The getNumMissed method returns the number of
35       questions missed.
36       @return The value in the numMissed field.
37    */
38
39    public int getNumMissed()
40    {
41       return numMissed;
42    }
43 }
```

```
1
2  public class Polymorphic
3  {
4     public static void main(String[] args)
5     {
6        // Create an array of GradedActivity references.
7        GradedActivity[] tests = new GradedActivity[3];
8
9        // The first test is a regular exam with a
10       // numeric score of 75.
11       tests[0] = new GradedActivity();
12       tests[0].setScore(95);
13
14       // The second test is a pass/fail test. The
15       // student missed 5 out of 20 questions, and
16       // the minimum passing grade is 60.
17       tests[1] = new PassFailExam(20, 5, 60);
18
```

```
er 10  Inheritance

19       // The third test is the final exam. There were
20       // 50 questions and the student missed 7.
21       tests[2] = new FinalExam(50, 7);
22
23       // Display the grades.
24       for (int i = 0; i < tests.length; i++)
25       {
26          System.out.println("Test " + (i + 1) + ": " +
27                             "score " + tests[i].getScore() +
28                             ", grade " + tests[i].getGrade());
29       }
30    }
31 }
```

26.

Output

Test 1 : score 95.0, grade A
Test 2 : score 75.0, grade P
Test 3 : score 86.0, grade B

27. instanceof operator
- To determine whether an object is an instance of a particular class

Example

```
FinalExam exam = new FinalExam(20, 2);
if (exam instanceof GradedActivity)
    System.out.println("Yes, exam is a GradedActivity");
else
    System.out.println("No, exam is not a GradedActivity");
```

28) Abstract Classes
- It is not instantiated, but other class extend it.

29) Abstract method
- it has no body and must be overridden in subclass.

30) Example of abstract method

```
public abstract void setValue(int value);
```

31) Also when a class contain an abstract method you cannot create instance of the class

32) Example

```
public abstract class Student
{
   private String name;        // Student name
   private String idNumber;    // Student ID
   private int yearAdmitted;   // Year admitted

   /**
      The constructor sets the student's name,
      ID number, and year admitted.
```

```
ter 10  Inheritance

      @param n The student's name.
      @param id The student's ID number.
      @param year The year the student was admitted.
   */

   public Student(String n, String id, int year)
   {
      name = n;
      idNumber = id;
      yearAdmitted = year;
   }

   /**
      The toString method returns a String containing
      the student's data.
      @return A reference to a String.
   */

   public String toString()
   {
      String str;

      str = "Name: " + name
         + "\nID Number: " + idNumber
         + "\nYear Admitted: " + yearAdmitted;
      return str;
   }

   /**
      The getRemainingHours method is abstract.
      It must be overridden in a subclass.
      @return The hours remaining for the student.
   */

   public abstract int getRemainingHours();
}
```

33.

```
/**
   This program demonstrates the CompSciStudent class.
*/

public class CompSciStudentDemo
{
   public static void main(String[] args)
   {
      // Create a CompSciStudent object.
      CompSciStudent csStudent =
            new CompSciStudent("Jennifer Haynes",
                               "167W98337", 2015);

      // Store values for math, CS, and gen ed hours.
      csStudent.setMathHours(12);
      csStudent.setCsHours(20);
      csStudent.setGenEdHours(40);

      // Display the student's data.
```

```
public class CompSciStudent extends Student
{
   // Required hours
   private final int MATH_HOURS = 20;   // Math hours
   private final int CS_HOURS = 43;     // Comp sci hours
   private final int GEN_ED_HOURS = 60; // Gen ed hours

   // Hours taken
   private int mathHours;  // Math hours taken
   private int csHours;    // Comp sci hours taken
   private int genEdHours; // General ed hours taken

   /**
      The Constructor sets the student's name,
      ID number, and the year admitted.
      @param n The student's name.
      @param id The student's ID number.
      @param year The year the student was admitted.
   */

   public CompSciStudent(String n, String id, int year)
   {
      super(n, id, year);
   }

   /**
      The setMathHours method sets the number of
      math hours taken.
      @param math The math hours taken.
   */

   public void setMathHours(int math)
   {
      mathHours = math;
   }

   /**
```

```
er 10  Inheritance

      The setCsHours method sets the number of
      computer science hours taken.
      @param cs The computer science hours taken.
   */

   public void setCsHours(int cs)
   {
      csHours = cs;
   }

   /**
      The setGenEdHours method sets the number of
      general ed hours taken.
      @param genEd The general ed hours taken.
   */

   public void setGenEdHours(int genEd)
   {
      genEdHours = genEd;
   }

   /**
      The getRemainingHours method returns the
      the number of hours remaining to be taken.
      @return The hours remaining for the student.
   */

   @Override
   public int getRemainingHours()
   {
      int reqHours,       // Total required hours
          remainingHours; // Remaining hours

      // Calculate the required hours.
      reqHours = MATH_HOURS + CS_HOURS + GEN_ED_HOURS;

      // Calculate the remaining hours.
      remainingHours = reqHours - (mathHours + csHours
                       + genEdHours);

      return remainingHours;
   }

   /**
      The toString method returns a string containing
      the student's data.
      @return A reference to a String.
   */
```

```
10.6 Abstract Classes and Abstract Methods

   @Override
   public String toString()
   {
      String str;

      str = super.toString() +
         "\nMath Hours Taken: " + mathHours +
         "\nComputer Science Hours Taken: " + csHours +
         "\nGeneral Ed Hours Taken: " + genEdHours;

      return str;
   }
}
```

```
hapter 10  Inheritance

      System.out.println(csStudent);

      // Display the number of remaining hours.
      System.out.println("Hours remaining: " +
                         csStudent.getRemainingHours());
   }
}
```

Program Output

```
Name: Jennifer Haynes
ID Number: 167W98337
Year Admitted: 2015
Major: Computer Science
Math Hours Taken: 12
Computer Science Hours Taken: 20
General Ed Hours Taken: 40
Hours remaining: 48
```

34 Interface specify behaviour for a class (make us know that the class is doing what we want it to do)

35. An interface can't be instantiated. It is implemented by other class. The class that use the interface must override the method that specified by the interface.

36. Example

```
public interface Displayable
{
   void display();
}
```

override

```
public class Person implements Displayable
{
   private String name;

   // Constructor
   public Person(String n)
   {
      name = n;
   }

   // display method
   public void display()
   {
      System.out.println("My name is " + name);
   }
}
```

```
public class InterfaceDemo
{
   public static void main(String[] args)
   {
      // Create an instance of the Person class.
      Person p = new Person("Antonio");

      // Call the object's display method.
      p.display();
   }
}
```

Output

My name is Antonio

37. Class that use the interface has the same method as interface which override the method of the class

38. Example of interface

```
public interface Relatable
{
   boolean equals(GradedActivity g);
   boolean isGreater(GradedActivity g);
   boolean isLess(GradedActivity g);
}
```

```
public class RelatableExams
{
   public static void main(String[] args)
   {
      // Exam #1 had 100 questions and the student
      // missed 20 questions.
      FinalExam3 exam1 = new FinalExam3(100, 20);

      // Exam #2 had 100 questions and the student
      // missed 30 questions.
      FinalExam3 exam2 = new FinalExam3(100, 30);

      // Display the exam scores.
      System.out.println("Exam 1: " + exam1.getScore());
      System.out.println("Exam 2: " + exam2.getScore());

      // Compare the exam scores.
      if (exam1.equals(exam2))
         System.out.println("The exam scores are equal.");

      if (exam1.isGreater(exam2))
         System.out.println("The Exam 1 score is the highest.");

      if (exam1.isLess(exam2))
         System.out.println("The Exam 1 score is the lowest.");
   }
}
```

39. field in interface

- All interface value are treated as final and static

Example

```
public interface Double{
        int FIELD1 = 1;
        int FIELD2 = 2;
        (Method header);
}
```

40. • Difference between abstract class and Interface

- A class can extend superclass where by interface can provide many interface implementation.

Example

public class MyClass implements Interface1, Interface2

41. Default method

- It is an interface method that has a body. the method is default.
- The class that use the interface can override the interface

Example

```
1 public interface Displayable
2 {
3    default void display()
4    {
5       System.out.println("This is the default display method.");
6    }
7 }
```

```
public class Person implements Displayable
{
   private String name;

   // Constructor
   public Person(String n)
   {
      name = n;
   }
}
```

```
6 public class InterfaceDemoDefaultMethod
7 {
8    public static void main(String[] args)
9    {
0       // Create an instance of the Person class.
1       Person p = new Person("Antonio");
2
3       // Call the object's display method.
4       p.display();
5    }
6 }
```

program output

This is the default method.

42. Using interface as reference variable for reference object

Example

File – interfaceExample.java

```
public interface interfaceExample{
        void displayMessage();
}
```

File – Example.java

```
public class Example implements interfaceExample{
        public mm(int i, int j){

        }
}
```

File – Main.java

```
public class Main {
        public static void main(String[] args){
                interfaceExample in = new Example(2,2);

        }
}
```

43. Anonymous Inner class extend another class or implement an interface

Example

File – IntCalculator.java

```
interface IntCalculator{
        int calculate(int number);
}
```

File – AnonymousClassDemo.java

```
public class AnonymousClassDemo{
        Public static void main(String[] args){
                int num;
```

```
        Scanner keyboard = new Scanner(System.in);
        IntCalculator square = new IntCalculator() {
            Public int calculate(int number){
                return number * number;
            }
        };
        System.out.print("Enter an integer ");
        num = keyboard.nextInt();
        System.out.println("The square is " +
                           square.calculate(num));
    }
}
```

Output

```
Enter an integer   5
The squar is 25
```

44. Example

```
class Machine{
    public void start {
        System.out.println("starting machine ....");
    }
}
public class App {
    public static void main(String[] args) {
        Machine machine1 = new Machine() {
            @Override public void start() {
                System.out.println("camera snapping");
            }
        };
    }
}
```

Output

```
Camera snapping
```

45. Functional Interface and Lambda Expression

- Creating an object that implement a functional interface

How

```
InterfaceName  variablename = parameter -> expression
```

Example

File - IntCalculator.java

```
interface IntCalculator {
        int calculate(int number);
}
```

File - LambdDemo.java

```
import java.util.Scanner
Public class LamdaDemo {
    public static void main(String[] args) {
        int num;
        Scanner keyboard = new Scanner(System.in);
        IntCalculator square = x -> x * x;

        System.out.println("Enter an interger");
        num = keyboard.nextInt();
        System.out.println("The square is " +
                           square.calculate(num));
    }
}
```

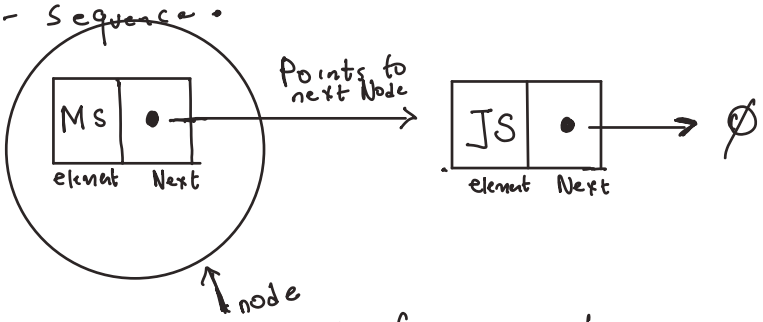
Output

```
Enter an interger 5 <- input
The square is 25
```

LINKED LIST

Singly Linked List

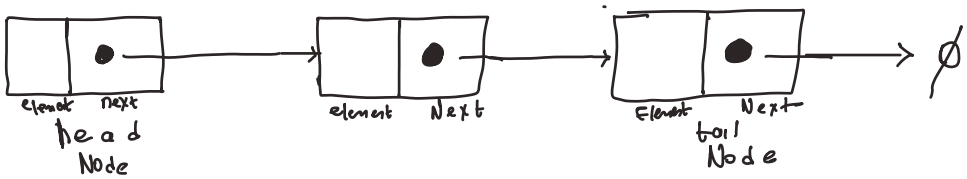
1. Linked List is a collection of nodes that form a linear sequence.



2. Link list is used instead of array because
   a) Array has fixed size.
   b) Insertion and deletion in Array interior position can be time consum if many element is shifted.

3. Singly Linked Lists

Example



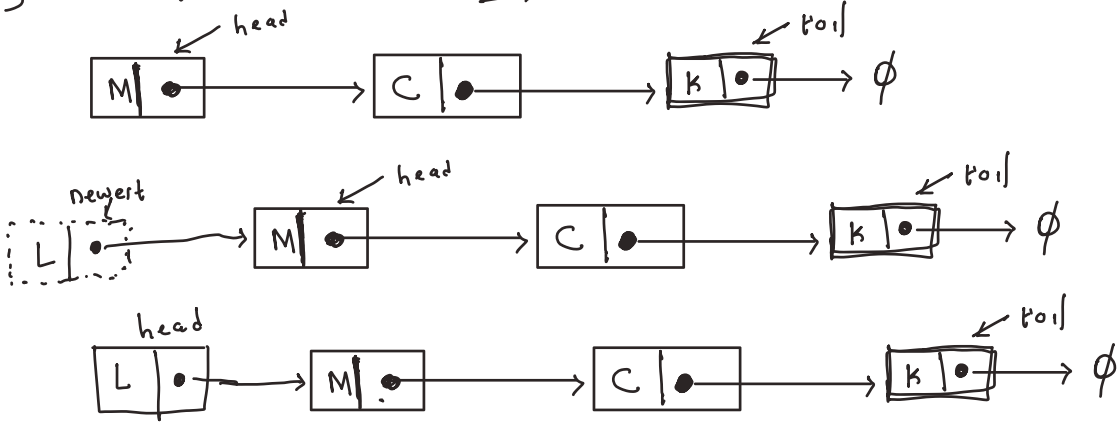
4. A singly Linked List must keep a reference to first node of the list which is the head. And also, an explicit reference to the tail

Ex

head =

tail =

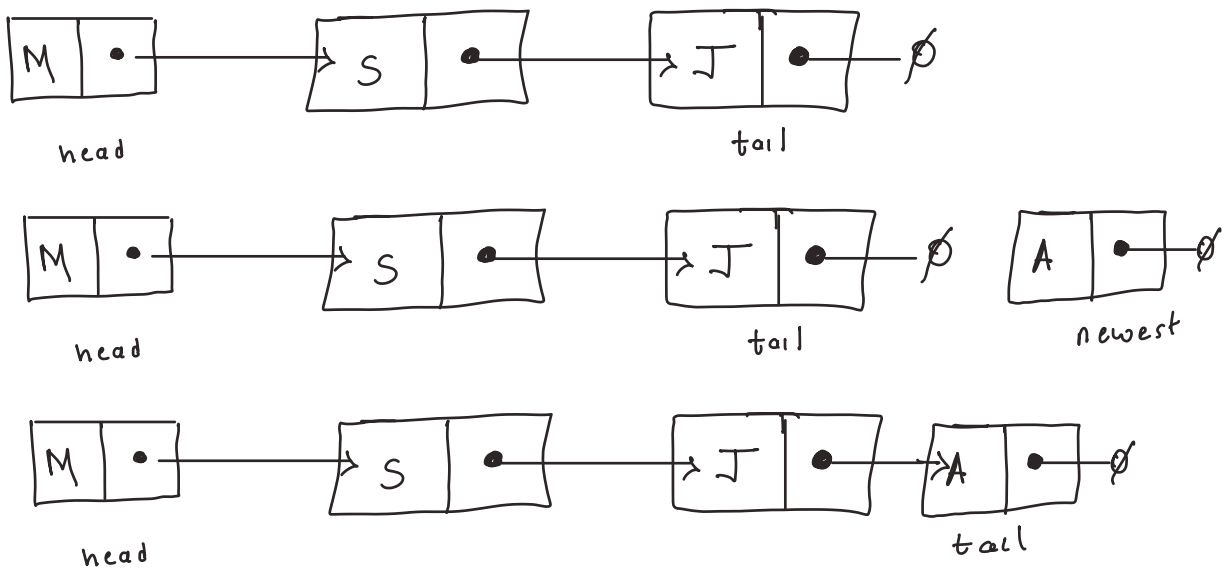
5. Inserting item at head of Singly Linked List



Algorithm

```
newest = Node(e);
newest.next = head;
head = newest
size++;
```

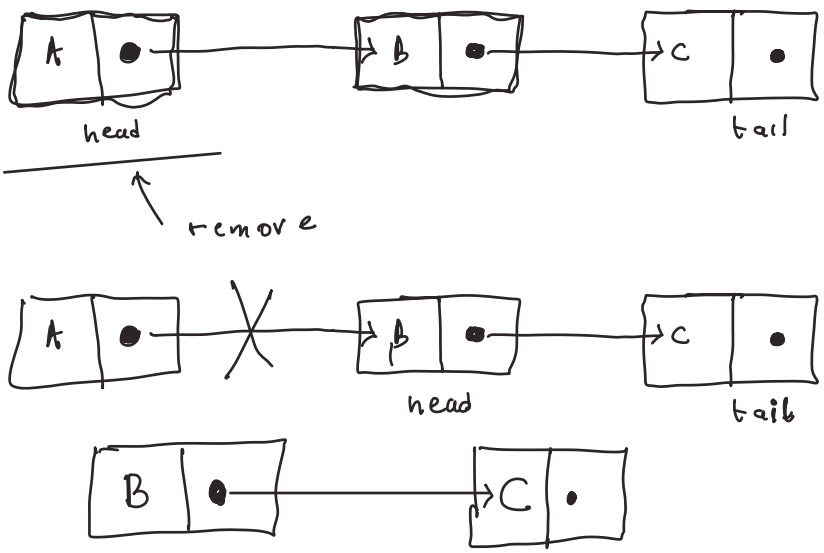
6) Inserting an element at the tail



Algorithm

```
Newest = Node(e);
Newest.next(null);
tail.next = Newest;
tail = newest;
size = size + 1;
```

7. Removing an element from singly Linked List (head)



Algorithm

```
if (size == 0) null then
head = head.next();
size--;
```

8) We sort is any. java generic framework that define a class with desired element type which

9) Example of Singly Linked List

```
public class SinglyLinkedList<E>{
    private static class Node<E>{
        private E element;
        private Node<E> next;
        public Node(E e, Node<E> n){
            element = e;
            next = n;
        }
        public E getElement(){return element;}
        public Node<E> getNext(){return next;}
        public void setNext(Node<E> n){next = n;}
    }
    private Node<E> head = null;
    private Node<E> tail = null;
```

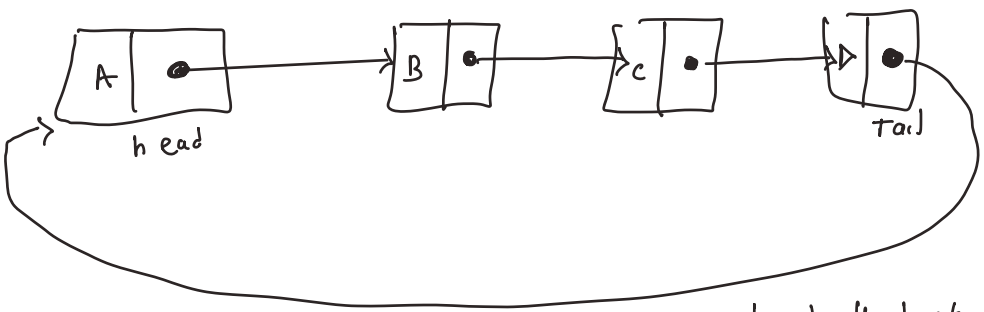
```
private int size = 0;
public SinglyLinkedList() {}
public int size() { return size; }
public boolean isEmpty() { return size == 0; }
public E first() {
    if (isEmpty()) return null;
    return head.getElement();
}
public E last() {
    if (isEmpty()) return null;
    return tail.getElement();
}
public void addFirst(E e) {
    head = new Node<>(e, head);
    if(size == 0)
        tail = head;
    size++;
}
public void addLast(E e) {
    Node<E> newest = new Node<>(e, null);
    if(size == 0){ head = newest;}
    else { tail.setNext = newest; }
    tail = newest;
    size++;
}
public E removeFirst() {
    if(isEmpty) return null;
    E answer = head.getElement();
    head = head.getNext();
    size--;
    if( size == 0) tail = null;
    return answer;
}
}
```

10. Circularly Linked Lists

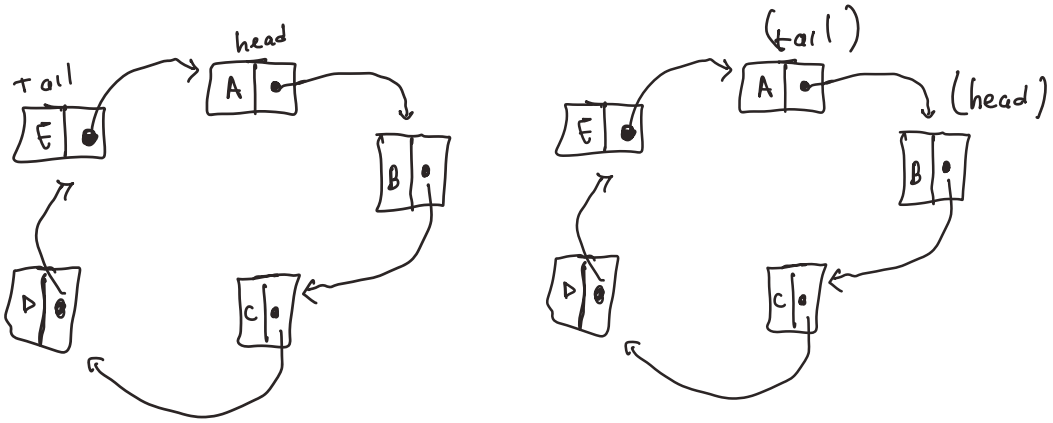
- It is having a cyclic order like multiplayer game where player A take turn, then player B, then player C then back to player A and continuing the cyclic part.

11. Design of a circular Linked List

- The tail node reference-to back to the head instead of null.



rotate(): ← move the first element to the back



{LAX, MSP, ATL, BOS}

{MSP, ATL, BOS, LAX}

Exam

```
Public class SinglyLinkedList<E>{
```

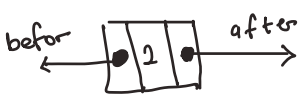
```
private static class Node<E>{
        private E element;
        private Node<E> next;
        public Node(E e, Node<E> n) {
              element = e;
              next = n;
        }
        public E getElement() { return element; }
        public Node<E> getNext() { return next; }
        public void setNext(Node<E> n) { next = n; }

}
private Node<E> tail = null;
private int size = 0;
private boolean isEmpty( ) { return size == 0 }
public E first( ) {
      if ( isEmpty( )) { return null; }
      return tail.getNext().getElement();

}
public E last( ) {
      if ( isEmpty( )) { return null ; }
      return last.getElement( );

}
public void rotate( ) {
      if ( tail != null )
            tail = tail.getNext( );

}
public void addFirst(E e ) {
      if ( size == 0 ) {
        tail = new Node<>(e, null);
        tail.setNext( tail );
      else {
        Node<> newest = new Node<>(e, tail.getNext());
        tail.setNext = newest;

      }
      size++;
}
```

```
public void addLast( E e ) {
        addFirst( e );
        tail = tail.getNext();

}

public E removeFirst( ) {
    if( isEmpty() ) return null;
     Node<E> head = tail.getNext();
     if( head == tail ) tail = null;
     else
         tail.getNext( head.getNext());

     return head.getElement();

}

}
```

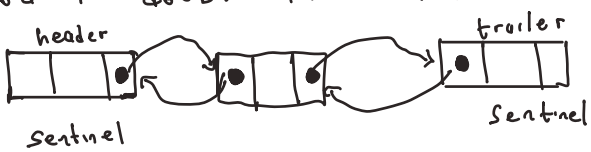
12. Double Linked Lists
    - This allow the deletion of a node easy if only given the reference to the node which is to be deleted. In a Singly linked list we do not know the node that come before the node to be deleted

    - A double linked list there is an explicit reference to to the node before and after it

Ex



13. It is good a double linked list to have a sentinels.



14. public class DoublyLinkedList<E> {
private static class Node<E> {
private E element;
Private Node<E> prev;
Private Node<E> next;
Public Node(E e, Node<E> p, Node<E> n){
ement = e;
rev = p;
ext = n;
}
public getElement( ) {
rn element;
}
public <E> getPrev( ) {
turn prev;
}
Public ode<E> getNext( ) {
eturn next;
}
public id setPrev(Node<E> p) {prev = p;}
Public oid setNext(Node<E> n) {next = n;}
}
private Node<E> head;
private Node<E> tail;
private int size = 0;
public DoublyLinkedList( ) {
head = new Node<>(null, null, null);
tail = new Node<>(null, head, null);
header.setNext(tail);
}
Public boolean isEmpty( ) {
return size == 0;
}
public int size( ) { return size; }

```
public E first( ) {
        if (isEmpty()) returnnull;
        return head.getNext().getElement( );
}
public E last( ) {
        if (isEmpty()) return null;
        return tail.getPrev().getElement( );
}
public void addFirst(E e){
        addBetween(e, header, header.getNext( ));
}
public void addLast(E e) {
        addBetween(e, trailer.getPrevl), trailer);
}
public E removeFirst( ) {
        if(isEmpty( )) return null;
        return remove(header.getNext( ));
}
public E removeLast( ) {
        if(isEmpty( )) return null;
        return remove(tail.getNext( ));
}
Private void addBetween(E e, Node<E>predecessor,
                        Node<E> sucessor){

        Node<E> newest : new Node<>(e, predecessor, sucessor);
        predecessor.setNext(newest);
        successor.setPrev(newest)e,
        Size++;
}
private E remove(Node<E> node) {
        Node<E> predecessor = node.getPrev( );
        Node<E> successor = node.getNext( );
        predecessor.getNext(successor);
        successor.setPrev(predecessor);
        size--;
        return node.getElement( );
```

```
    }
}
```

15. Equivalence testing in array

- Arrays.equals(a,b);
  ↑ to compare one dimensional array

Example

```
int[] array1 = {1,2,3};
int[] array2 = {2,3,4};

boolean value=Arrays.equals(array1, array2)
System.out.println(value);
```

Output
false

- Arrays.deepEquals(a,b):
  ↑ use for two dimensional array

Example

```
import java.util.Arrays;
int[][]array1 = {{10,20}, {20,30}}
int[][]array2 = {{10,30}, {20,30}}

boolean value=Arrays.deepEquals(array1, array2)
System.out.println(value);
```

Output
false

16. Equivalence testing with linked lists

```
public boolean equals(Object o){
    if(o==null) return false;
    If (getClass() != o.getClass()) return false;
    SinglyLinkedList other = (SinglyLinkedList) o;
    if(size != other.size) return false;
```

if the list is null then it not equal

Both must be SinglyList for it to be consider equal

set the singleList object

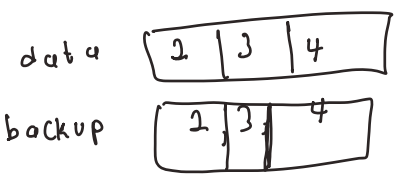
```
Node walkA = head;
Node walkB = other.head;
while (walkA != null) {
    if (!walkA.getElement().equals(walkB.getElement()))
        return false;
    walkA = walkA.getNext();
    walkB = walkB.getNext();
}
```

this test every Node each to see if it is equal

17. Shallow copy created a new array whose cell refer to same object referenced by first array.

Example

```
int[] data = { 2, 3, 4 }
int[] backup = data.clone();
```




```
data[0] = 5;
System.out.println(backUp[1]);
```

doesn't affect the other array

output

3

16. Deep copy is cloning individual element

Example

```
person[] guests = new Person[contacts.length];
for(int k=0; k < conctacts.length; k++)
```

```
guests[k] = (Person) contacts[k].clone();
```

cloning individual item.

19.