[Cheat Sheet] - Java

Basic Java structure:

```
class CodeName
{
    public static void main(String[] args){
    int a = 1;
    float b = 1.0;
    long d = 1.1;
    double e = 1.2;
    char f = "A";
    String g = "String";
    boolean h = true;

    System.out.println("...")//Print a line
    }
}
```

Strings:

- You can't compare strings by variable names
  - Compare the memory addresses (class 09/21)
- .equals(otherString) - compares the strings, not the addresses
- .atChar(index) - reads the char at index
- .substring(x, y) - gets the chars from x to y
- .toUpperCase() - transforms the char/string into upper case
- .toLowerCase() - transforms the char/string into lower case

Arrays:

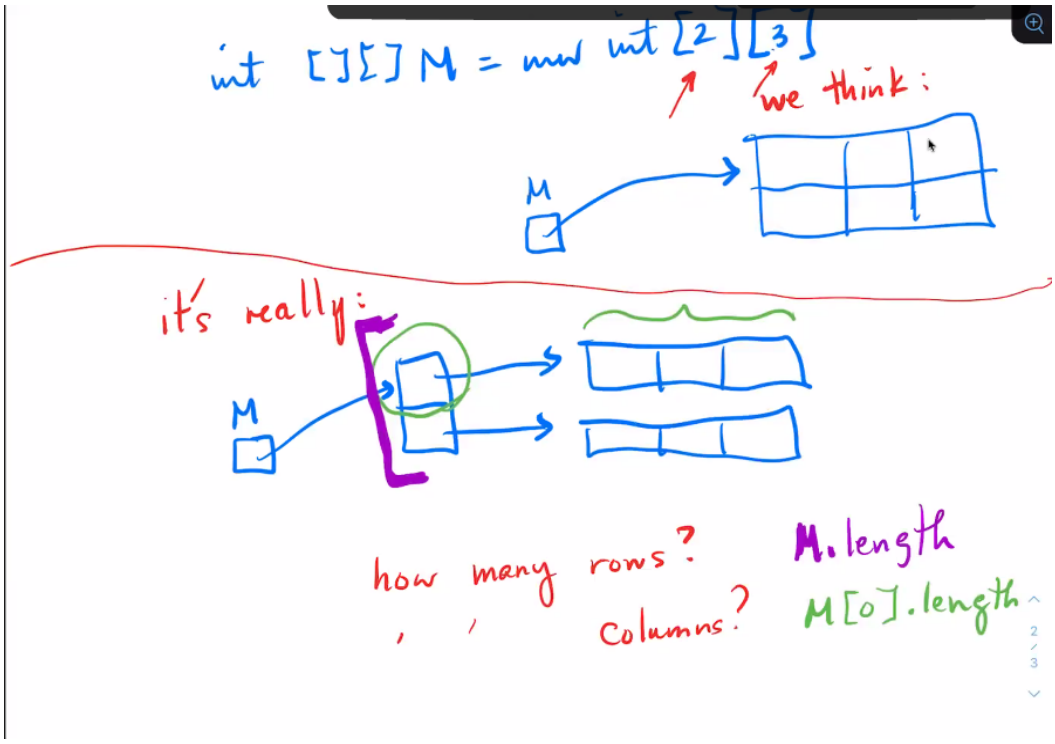
- **Single**
  - to declare an array, you specify the datatype, put brackets, and give a name for the variable

```
int[] arr = new int[5]; //first declare a array variable, then give its size
int[] arr2 = {1,2,3,4,5}; //hard code the values
```

```
arr[i] = x; //assign a value for that pointer

arr.length //Access arr's length
```

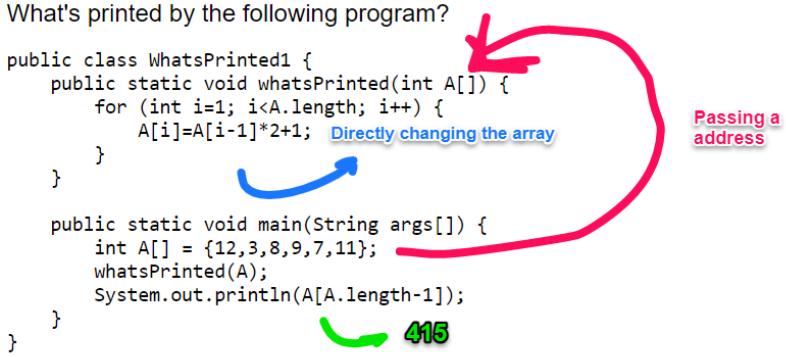
- **Multidimensional:**
  - array that contain other arrays
  - To create multidimensional arrays, place each array within its own set of square brackets

```
int[/*ROW*/][/*COL*/] sample = { {1, 2, 3}, {4, 5, 6} };
int x = sample[0][0];
System.out.println(x)//Output = 1
```



- **Dealing with Arrays**
- you are dealing with **Memory Addresses**

- Using a single array in all program
  - What's printed by the following program?



  -
  -

Functions/Methods:

- If you are passing something into a function, you are passing a **copy**
- **Method overload**
  - Create 2 functions with the same name, but pass different parameters

```
class MethodExample{
    //This is a method
    static void sayHello(){
        System.out.println("Hello");
    }
    //method with parameter
    static void nameHello(String name){ //TypeDef variableName
        System.out.println("Hello " + name);
    }

    static void nameHello(String name1, String name2){//Method Overload
    System.out.println("Hello " + name + " and " + name2);
    }
    //main function
    public static void main (String[] args){
```

```
        sayHello();
        nameHello("Gabriel");
    }
}
```

Loops:

- **While**
  - Same as c

```
while (condition){
...
}
```

- **For loops**
  - Same as c

```
for(int begin = ?; begin < end; begin++){
    ...
}
```

- **Enhanced For loop or "for each"**
  - used to traverse elements in arrays.

```
int arr[] = new int[5];
int x = 0;
for(x : arr){
    System.out.println(x);
}
```

- **Do while**
  - Same as C

```
do{
...
}while(condition);
```

```java
public class Cars {
  public static void main(String []args) {
    int numCars = 0;
    int numLux = 0;
    int sumLux = 0;

    Scanner in = new Scanner(System.in);
    // cur = get the first thing
    int cur = in.nextInt();
    while (cur >= 0) {
      numCars++;

      if (cur > 50000) {
        numLux++;
        sumLux += cur;
      }

      //   get the next thing
      cur = in.nextInt();
    }

    if (numCars == 0) {
      System.out.println("No cars entered");
    } else if (numLux == 0) {
      System.out.println("No luxury");
    } else {
      System.out.println("Average lux price = " + sumLux / numLux);
    }
  }
}
```

Switch case

- same as in C

```
switch (operation){
    case 1:
    ...
    break;

    case 2:
    ...
    break;
}
```

- Switch Statement
  - multiple comma-separated values per case and returns a value for the whole switch-case block.

```
int day = 11;
String dayType  = switch(day) {
         case 1, 2, 3, 4, 5 -> "Working day"; // -> instead of :
         case 6, 7 -> "Weekend";
         default -> "Invalid day";
};
        System.out.println(dayType);
```

Random Numbers:

```
import java.util.Random;//or  import java.util.*
Random rand = new Random();
```

- rand.nextInt() - returns a random integer
- rand.nextInt(MAX) - returns a random integer from 0 to MAX
  - rand.nextInt(MAX) + MIN_VALUE //
- rand.nextDouble()

Arrays Library:

- import java.util.Arrays
  - Arrays.toString(arr) - prints a list of elements in the array
  - Arrays.asList(arr).contains("a"); - finds the element "a" in array "arr"
- import java.util.ArrayList
  - arr.add(element) - adds an element to array
  - arr.get(index) - gets element from array's index
  - arr.set(index, element) - sets the arrays index element to the parameter

User Input:

-

```
import java.util.Scanner; // import scanner
import java.util.* //importas all util library

Scanner x = new Scanner(System.in); // make x a scanner to get user input
and assign it to; System.in = keyboard

E.g:
int y = x.nextInt(); // Gets a int from the user and assigns to y
```

- List of scannable things:
    - Read a byte - nextByte()
      Read a short - nextShort()
      Read an int - nextInt()
      Read a long - nextLong()
      Read a float - nextFloat()
      Read a double - nextDouble()
      Read a boolean - nextBoolean()
      Read a complete line - nextLine()
      Read a word - next()
- 

Conversions:

- https://www.javatpoint.com/java-string-to-int

Recursion:

- When a method calls itself
- Generally used with if/else to avoid infinite loop

- ## Java Recursion Example 3: Factorial Number

```
public class RecursionExample3 {
   static int factorial(int n){
       if (n == 1)
         return 1;
       else
         return(n * factorial(n-1));
   }

public static void main(String[] args) {
System.out.println("Factorial of 5 is: "+factorial(5));
}
}
```

Output:

```
Factorial of 5 is: 120
```

Working of above program:

```
factorial(5)
   factorial(4)
      factorial(3)
         factorial(2)
            factorial(1)
               return 1
            return 2*1 = 2
         return 3*2 = 6
      return 4*6 = 24
   return 5*24 = 120
```

Another One

1) base case
2) recursive step

```
1  public class SimpleRecur2 {
2      public static int f(int x) {
3          if (x==1) {
4              return 2;
5          } else {
6              return 2*f(x-1)+1;
7          }
8      }
```

Base case

Recursive Step

Files:

```
import java.io.*;

class FILENAME{
    public static void main(String[] args)
        throws FileNotFoundException{ //NEEDED!!!
        File file = new File(FILENAME2);
        Scanner input = new Scanner(file);
        //OR
```

```
        Scanner input2 = new Scanner(new File(FILENAME2));//if you are going
to read the file only once
    }
}
```

- Most used to read/analyze files from your OS.

- Create a `File` object to get info about a file on your drive.
  - (This doesn't actually create a new file on the hard disk.)

```
File f = new File("example.txt");
if (f.exists() && f.length() > 1000) {
    f.delete();
}
```

Method name	Description
`canRead()`	returns whether file is able to be read
`delete()`	removes file from disk
`exists()`	whether this file exists on disk
`getName()`	returns file's name
`length()`	returns number of bytes in file
`renameTo(`*file*`)`	changes name of file

- **absolute path**: specifies a drive or a top `"/"` folder

  `C:/Documents/smith/hw6/input/data.csv`

  - Windows can also use backslashes to separate folders.

- **relative path**: does not specify any top-level folder

```
names.dat
input/kinglear.txt
```

  - Assumed to be relative to the *current directory*:

```
Scanner input = new Scanner(new
File("data/readme.txt"));
```

If our program is in `H:/hw6`,
`Scanner` will look for `H:/hw6/data/readme.txt`

```
Who lives in a pineapple under the sea?
Spongebob Squarepants
Absorbent and yellow and porous is he
Spongebob Squarepants
```

lyrics.txt

```
import java.util.Scanner;
import java.io.*;

public class ReadFileTst {
  public static void main(String []args)
      throws FileNotFoundException {
    String filename = "lyrics.txt";
    int NUM_WORDS = 19;

    Scanner in = new Scanner(new File(filename));
    for (int i = 0; i < NUM_WORDS; i++) {
      String word = in.next();
      System.out.println(word);
    }
  }
}
```

ReadFileTst.java

`Scanner` tests for valid input

Method	Description
`hasNext()`	returns `true` if there is a next token
`hasNextInt()`	returns `true` if there is a next token and it can be read as an `int`
`hasNextDouble()`	returns `true` if there is a next token and it can be read as a `double`

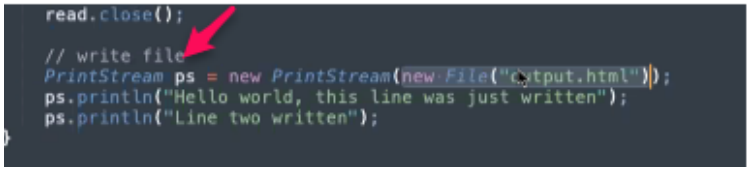
- These methods of the `Scanner` do not consume input; they just give information about what the next token will be.
  - Useful to see what input is coming, and to avoid crashes.

```java
import java.io.*;

public class OutToFileExample {
  public static void main(String []args) throws FileNotFoundException {
    //    System.out.println("There's always money in the banana stand.");
    PrintStream outToFile =
        new PrintStream(new File("deepMessage.txt"));
    outToFile.println("There's always money in the banana stand.");
  }
}

```

U:--- OutToFileExample.java All L7 (Java//lw , +3 Abbrev)

```
bash-3.2$ l
bash: l: command not found
bash-3.2$ more deepMessage.txt
There's always money in the banana stand.
bash-3.2$
```

modifies an file(must be on dir)



"Today is tomorrow's yesterday" - logic

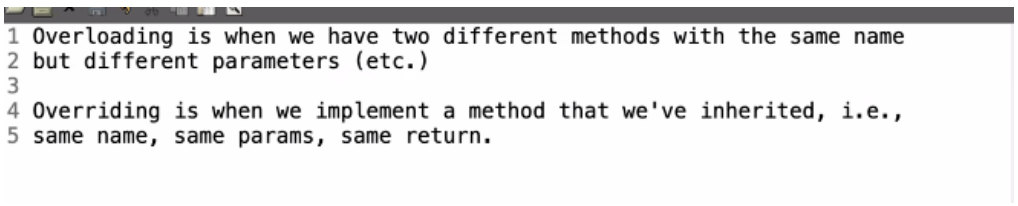
Classes and Objects:

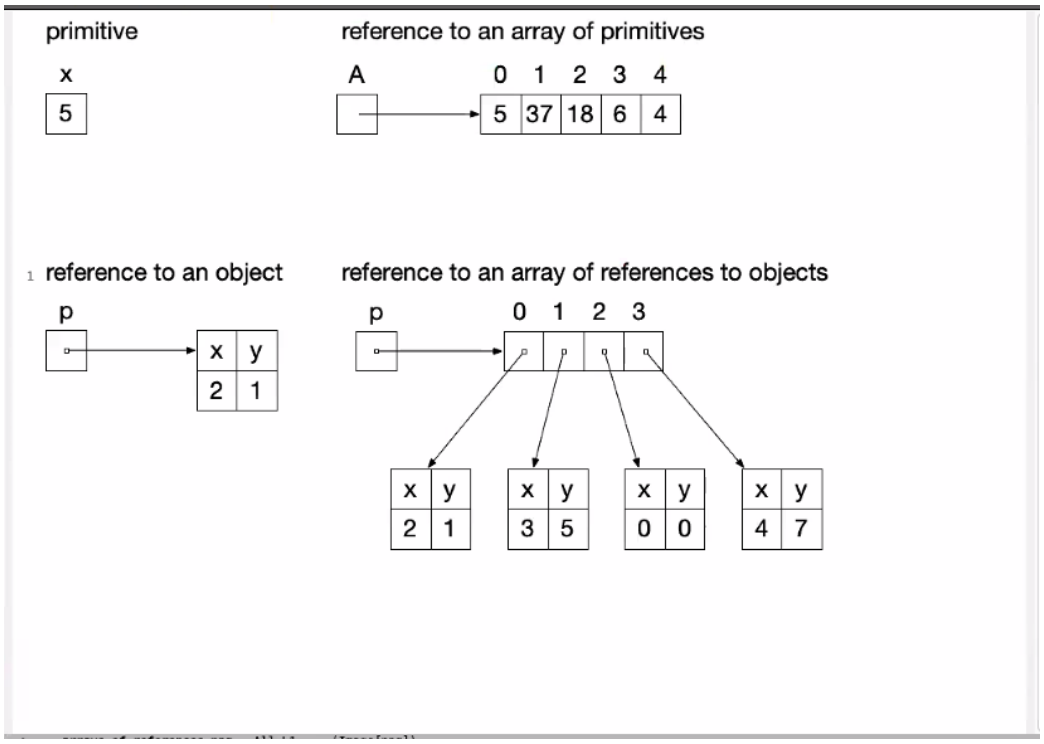
- **Object**
  - "Attributes describe the object's current state, and what the object is capable of doing is demonstrated through the object's behavior."
  - object is an instance of a class
  - to be modified outside the main function, it needs a return statement
- **Class**
  - can be described as blueprints, descriptions, or definitions for an object
  - used to define **attributes** and **behavior**

        - **attributes** are basically variables within a class.
        - Methods define **behavior**
        - You can also create a method that takes some data, called **parameters**
- **Constructor**
    - special method called when we create a new operator

Class + Main Function example

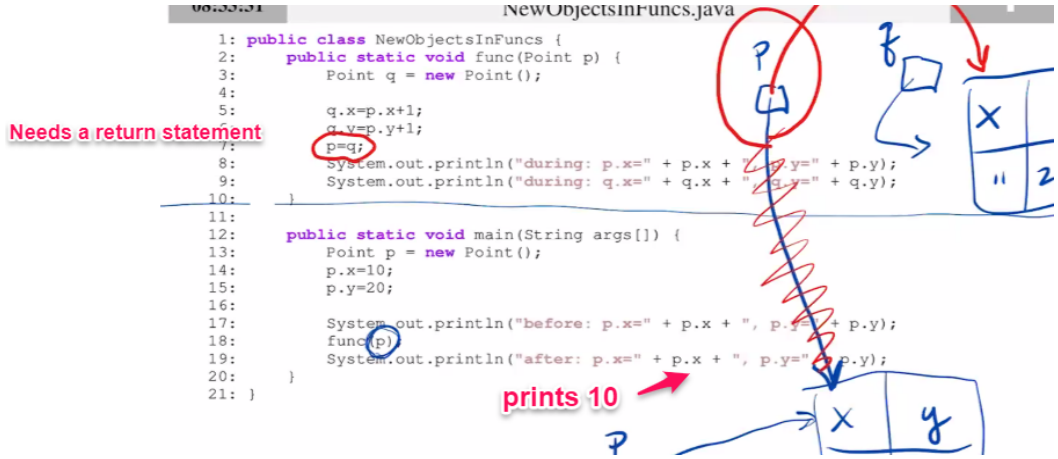
```
//Classes
public void Class{
    int a;
    int b;
    String c;
    double d[] = new double[/*size*/];

    public void print(){//Public --> access everywhere
        System.out.println("Something");
    }
}


//Main Code
class MainCode{
    Class x = new Class();
    x.a = 1;
    x.b = 2;
    x.c = "Something";
    x.d[0] = 1.1;

    x.print();
}
```





Constructor Examples:

```
public class OOP {
        int ID;
        int AGE;
        String NAME;
        public void printInfo() { //prints in a format(oop.printInfo)
                System.out.println("ID: "+ ID);
                System.out.println("AGE: "+ AGE);
                System.out.println("NAME: "+ NAME);
        }
        //initializes an OOP object with those informations(new
OOP(1,2,"name"))
        //same name as class
        public OOP(int initialID, int initialAGE, String initialNAME) {
                ID = initialID;
                AGE = initialAGE;
                NAME = initialNAME;
        }
}
```



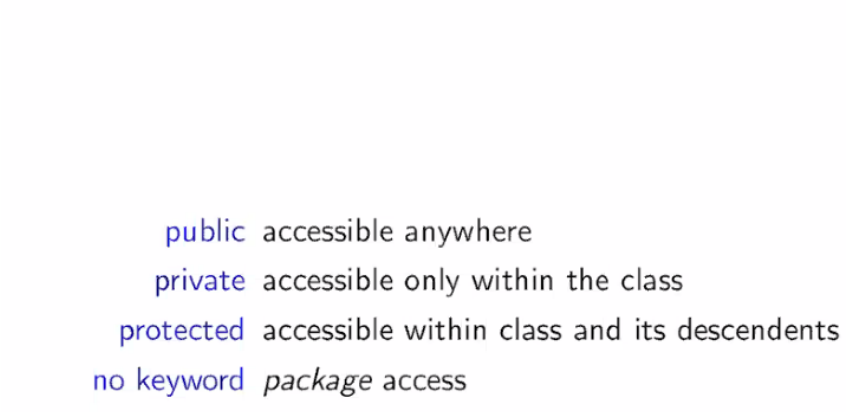

```java
public class Point {
  int x;
  int y;

  /* constructor. special method called only when
   *      we create a Point object using "new" */
  public Point(int initialX, int initialY) {
    x = initialX;
    y = initialY;
  }

  // public String toString(){
  //   return "(" + x + "," + y + ")";
  // }
}
```

- **Access Modifiers**
  - **public**: The class is accessible by any other class.
  - **default**: The class is accessible only by classes in the same package.
  - **protected**: Provides the same access as the default access modifier, with the addition that subclasses can access protected methods and variables of the superclass (Subclasses and superclasses are covered in upcoming lessons).
  - **private**: Accessible only within the declared class itself.

- Remaining Java Protection Levels



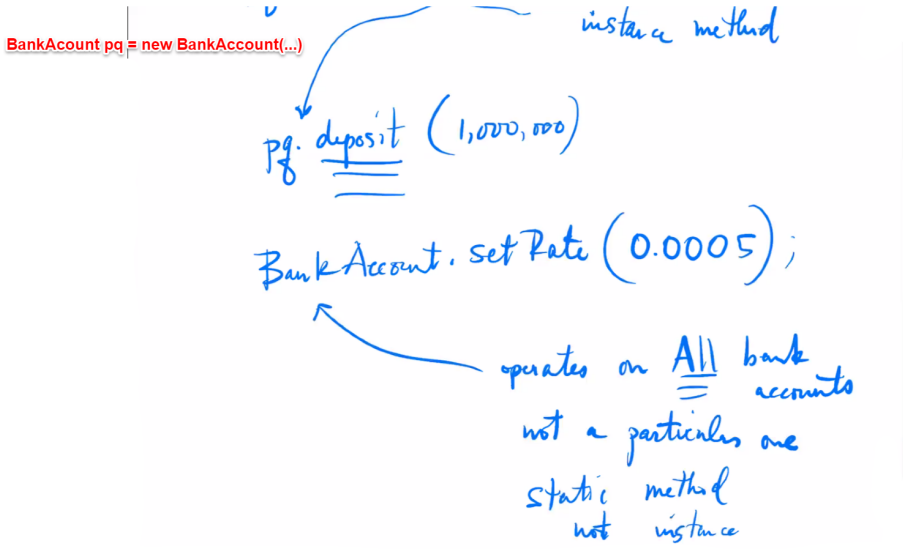
- **Getters and Setters**
  - **getter** method returns the value of the attribute.
  - **setter** method takes a parameter and assigns it to the attribute.
  - ```
    public int getNum() {
        return enumerator;
    }

    public int getDenom() {
        return denominator;
    }

    public void setNum(int n) {
        enumerator = n;
    }

    public void setDenom(int d) {
        denominator = d;
    }
    ```
    returns/reads a private variable
    assigns a value to variable
- Static method VS Instance method
  - Instance -> operates in a certain object
    ```

-



Inheritance Vs Composition:

- Inheritance (**is a**)
  -

Improved `Secretary` class

```
// A class to represent secretaries
public class Secretary extends Employee
{
    public void takeDictation(String text) {
        System.out.println("Taking dictation of text: "
                           + text);
    }
}
```

Different method that is not in the "main" class

-
```java
public class Point {
  protected int x;
  protected int y;

  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public void move(int dx, int dy) {
    x += dx;
    y += dy;
  }
```

```java
public class Point3D extends Point {
  int z;

  public Point3D(int x, int y, int z) {
    this.x = x;
    this.y = y;
    this.z = z;
  }

  public Point3D(int x, int y, int z) {
    super(x, y);
    this.z = z;
  }
```

  - Super: used in the extended class to "import" the Super Class' statements
  - The parent class **DOESN'T** have a constructor, only methods
  - If the code doesn't compile(constructor OOP in class OOP cannot be applied to given types;)
    - Create a no value constructor in OOP
    -
```java
public class OOP2 extends OOP{
    int y;

    public OOP2(int x, int y, String name){
        super (x, name); super uses the param above
        this.y = y;
    }


    public void vroom(){
        System.out.println("Vroom 2");
    }
}
```

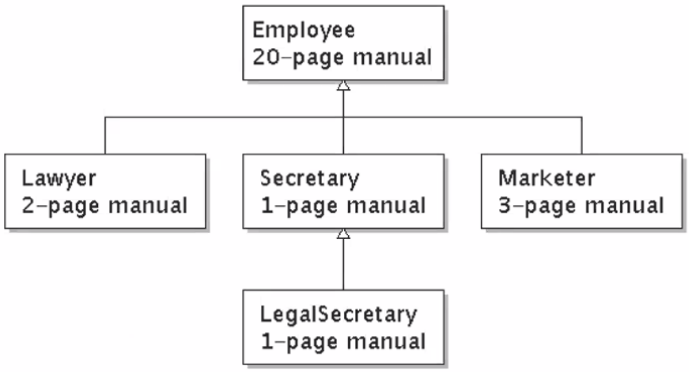
```java
public class OOP{
    int x;
    String name;

    public OOP(int x, String name){
        this.x = x;
        this.name = name;
    }

    public OOP(){
        this.x = 0;
        this.name = "";
    }
```

    -
```java
OOP op3 = new OOP2(5,3,"name3");//a parent can be a child
op3.vroom(); //method from OOP2 works
//OOP2 op4 = new OOP(5,"name4"); //a child cant be a parent
```

    -

- Composition(**has a**)

- The smaller manual adds some rules and also changes (read: overrides) some rules from the large manual (e.g., "use the pink form instead of the yellow form")



- **Override**
  - A method in the child function that has the same name as a method in the parent function
  - no need for the @Override
  - ```
    @Override
    public double getSalary() {
        return 45000.0;          // $45,000.00 / year
    }
    ```

- Polymorphism
- Parent Class Child Class

```
OOP1 op3 = new OOP2(369, "Secretary", 3999.99, "Chad", true);
```

- The ability for the same code to be used with several different types of objects and behave differently depending on the type of object used.
- op3 has 2 types
```

    - Static - compiler uses to determine if statements are legal; any method called with the person variable must be declared in the Employee class
    - Dynamic - Java virtual machine uses to execute code when the program is run; Any method called with the person variable will execute the version of that method defined in the Lawyer class
- op3 can't use methods from OOP2
    - We can use ((OOP2)op3).method() --> method being only present in OOP2
- Assume that the following four classes have been declared:

```
public class Foo {
    public void method1() {
        System.out.println("foo 1");
    }

    public void method2() {
        System.out.println("foo 2");
    }

    public String toString() {
        return "foo";
    }
}

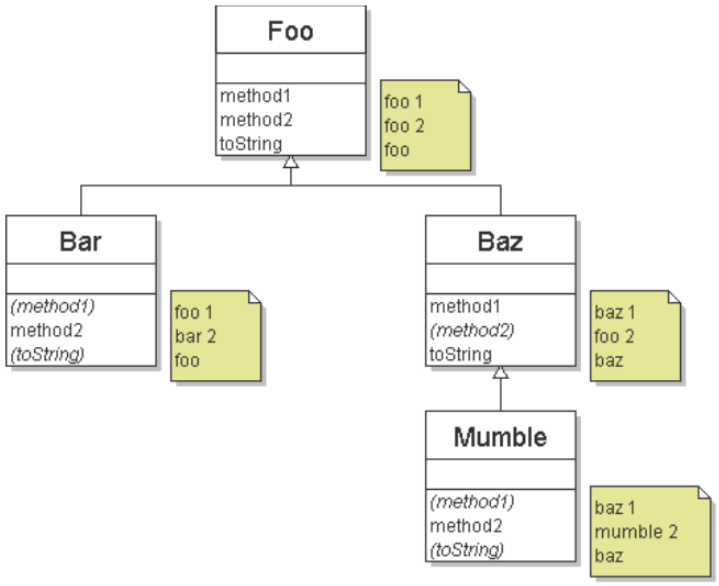
public class Bar extends Foo {
    public void method2() {
        System.out.println("bar 2");
    }
}
```

```
public class Baz extends Foo {
    public void method1() {
        System.out.println("baz 1");
    }

    public String toString() {
        return "baz";
    }
}

public class Mumble extends Baz {
    public void method2() {
        System.out.println("mumble 2");
    }
}
```

- Diagramming polymorphic code




```
Employee team[] = new Employee[3];

team[0] = new Marketer();
team[1] = new Lawyer();
team[2] = new LegalSecretary();

for (int i=0; i<team.length; i++)
    System.out.println(team[i].getSalary());
```

Variable Shadowing:
Something to avoid!!

```
public class A {
  int x = 1;
  int method() { return 1; }
}
public class B extends A {
  int x = 2;
  int method() { return 2; }
}
```

```
A a1 = new A();
A a2 = new B();

System.out.println(a1.method());
// prints 1
System.out.println(a2.method());
// prints 2

System.out.println(a1.x);
// prints 1
System.out.println(a2.x);
// prints 1 still!
// because reference a2 has
// compile-time type A.
```

Packages

- group of classes and constants - bundle
- share between devs
- The first lines of code in your file:

```
/* some comments here are ok */
package packagename;

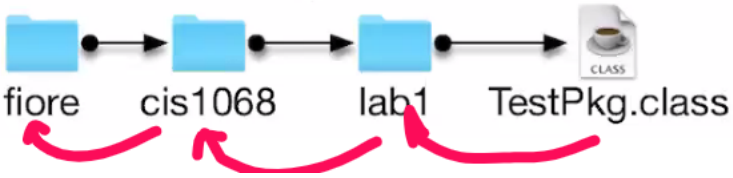
public class Whatever {
    ...
```

- It matters. Suppose that we have:

```
package fiore.cis1068.lab1;

public class TestPkg {
    public static void main(String args[]) {
        System.out.println("Did this work?");
    }
}
```

Must place resulting class file in:



-
- Name Clash
  - There can only be one name
-
  - What happens if we don't use the package statement?
  - Class becomes part of the **default package**
    - all of the classes in the current directory
-

Exception Handling:

- **`throws` clause**: Keywords on a method's header that state that it may generate an exception (and will not handle it).

- Syntax:

```
public static type name(params) throws type {
```

  - Example:

```
public class ReadFile {
    public static void main(String[] args)
            throws FileNotFoundException {
```

  - Like saying, *"I hereby announce that this method might throw an exception, and I accept the consequences if this happens."*

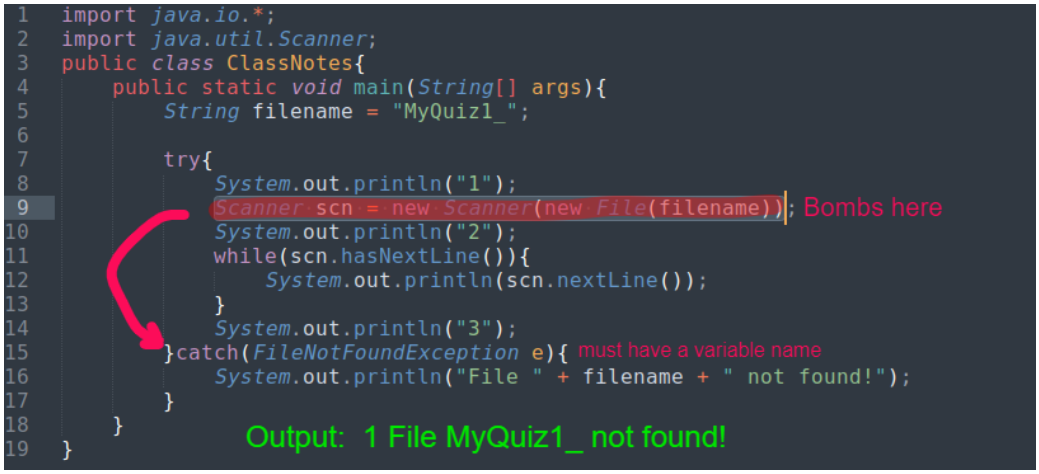
- it can be also used in methods(throw)

```
public Question(int points, int difficulty,
         int answerSpace, String questionText){
                this.points = points;
                //Checks if difficulty is valid
                if(difficulty>=MIN_DIFFICULTY && difficulty<=MAX_DIFFICULTY)
{
                        this.difficulty = difficulty;
                }else{
                        throw new IllegalArgumentException("Difficulty not
in range");//exception
                }
                this.answerSpace = answerSpace;
                this.questionText = questionText;
        }
```

- IllegalArgumentException - throws an error if the input is not what you expect
- ArithmeticException - zero division or invalid input in equations

Try/catch exception handling

```
try {
   /* stuff you want to do.
    * no error handling code */
} catch (SomeKindOfException) {
   /* exception handling code */
} finally{
   //Executes anyway
   }
```




```
import java.io.*;
import java.util.Scanner;
public class ClassNotes{
    public static void main(String[] args){
        String filename = "MyQuiz1_";

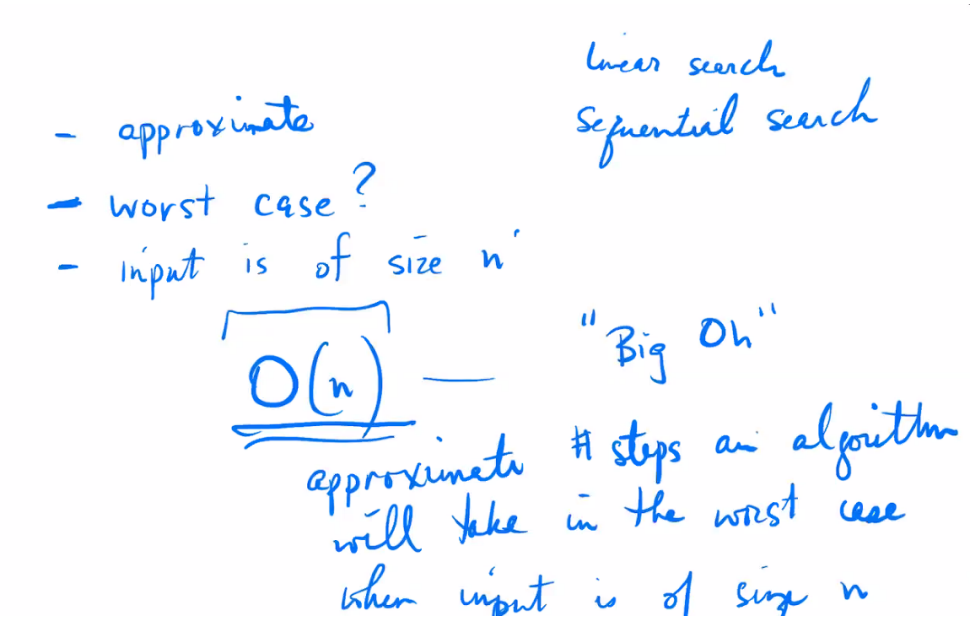
        try{
            System.out.println("1");
            Scanner scn = new Scanner(new File(filename));
            System.out.println("2");
            while(scn.hasNextLine()){
                System.out.println(scn.nextLine());
            }
            System.out.println("3");
        }catch(FileNotFoundException e){
            System.out.println("File " + filename + " not found!");
        }
    }
}
```

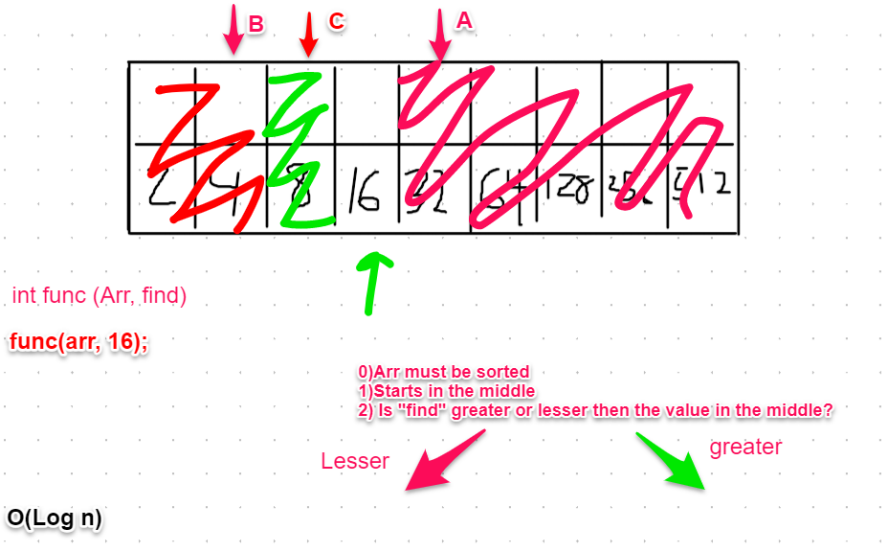
JUnit Testing(Eclipse IDE):

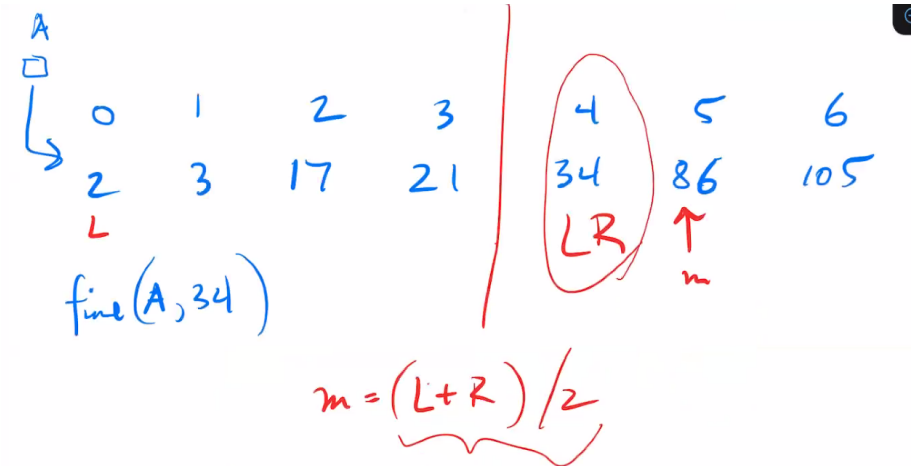
```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class TESTFILENAME{
    @Test
    void testName(){
       assertTrue(FILENAME.method(x));
       assertFalse(FILENAME.method(x));
       assertEquals(FILENAME.method(x, y);
    }
}
```

Measuring efficiency of Algorithm

- How many steps the algorithm takes in the worst case scenario?
- 

- **Binary Search**
  - 

- 

-
- if what we are looking for is grater than middle; ignore left(L = m+1)
- if what we are looking for is less than middle; ignore right(R = m-1)
- when left is grater then right, the item is not in the array

```java
public class ClassNotes{
    public static void main(String[] args){
        int arr[] = {2,4,6,7,8,9,10,22};
        System.out.println(find(arr, 6));

    }


    static int find(int[] arr, int item){
        int l = 0;
        int r = arr.length-1;
        //System.out.println(m + " " + l + " " + r);
        while(l<=r){
            int m = (l+r)/2;
            if(arr[m] == item){
                return m;
            }else if(item > arr[m]){
                l = m+1;
            }else{
                r = m-1;
            }
        }
        return -1;
    }
}
```

Output: 2