We gonna be using open source JDK, the version we Using is created by Amazon, and its called amazon corretto. We be using jdk version 11. version 11 is a LTS version I.e is Long term support version.
You need java so
1. search amazon corretto
jdk 15 x86,x64 <!-- x86 is a 32 bit version
2. Intellij

Done.
Now, we have to associate jdk 15 with intellij, basically thats telling intellij where jdk is stored on the computer.

SDK, Software Development Kit

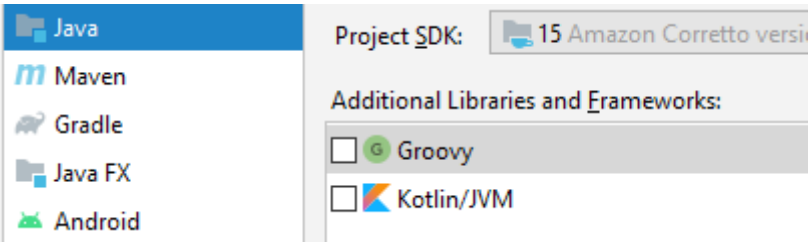
The JDK, is effectively a Software Development kit, or SDK.

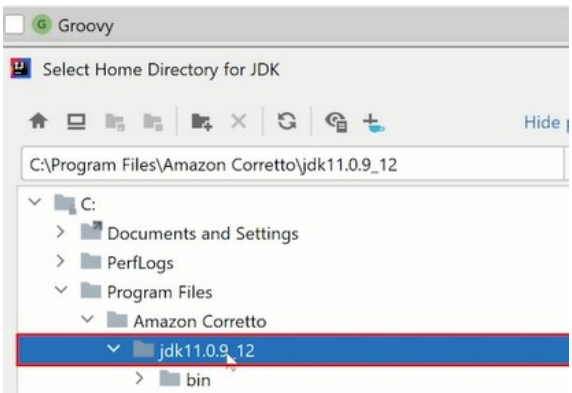
Whatever you call it, it contains the tools you need to write programs.

The Java Development Kit includes the tools that enables the computer to understand your Java code, and to execute it.

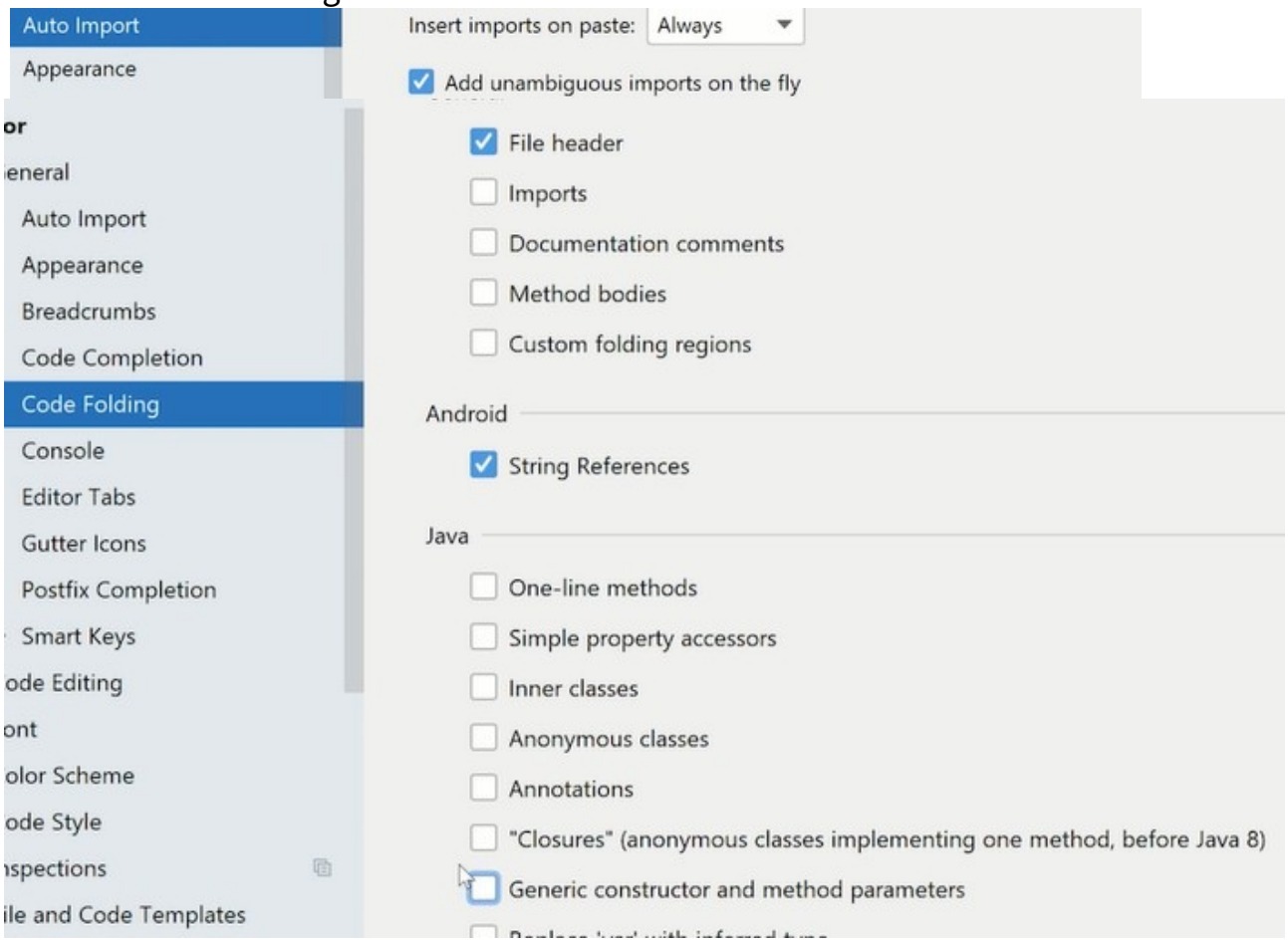
It also has a debugger, and we'll be seeing what that is, and how to use it, when we've written a program to debug.

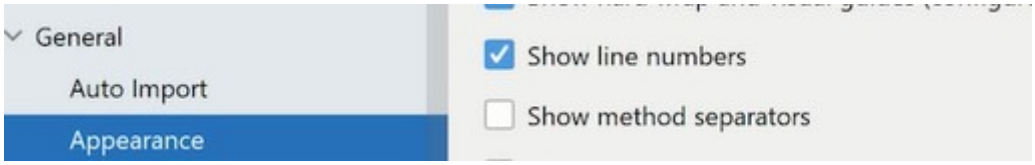
If u use a version control sys from git, so git from vcs will fetch ur fetch ur project from vcs.



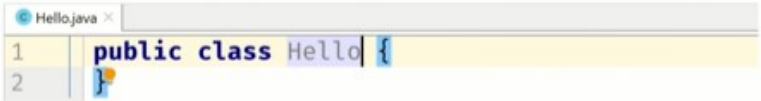


Goto
Customize > All settings > Editor > General





Hello World Project



Java programs (and most other programming languages) have keywords. Each has a specific meaning and sometimes they need to be used in specific orders.

You write Java programs by following a specific set of rules, using a combination of these keywords and other things you will see which collectively form a Java program.

NOTE: keywords are case sensitive - public and Public and even PUBLIC are different things.

public and **class** are two java keywords - they have a specific meaning which we will find out more about moving forward.

Hello world

The **public** Java keyword is an access modifier. An access modifier allows us to define the scope or how other parts of your code or even some else's code can access this code. More on that later.

For now, we will use the public access modifier to give full access. We'll come back to access modifiers once we get further into the course.

Defining a class. The **class** keyword is used to define a class with the name following the keyword - Hello in this case and left and right curly braces to define the class block.

To define a class requires an optional access modifier, followed by class, followed by the left and right curly braces.

The left and right curly braces are defining the class body - anything in between them is "part" of this class. We can have data and code as you will see as we progress.

So that's our first class defined.

Defining the Main Method

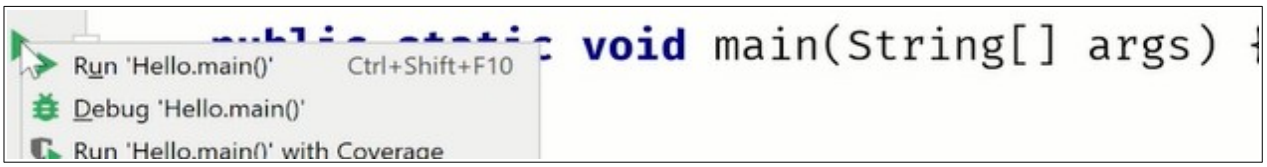
What is a method? It's a collection of statements (one or more) that performs an operation. We'll be using a special method called the main method that Java looks for when running a program. It's the entry point of any Java code.

void is yet another java keyword used to indicate that the method will not return anything

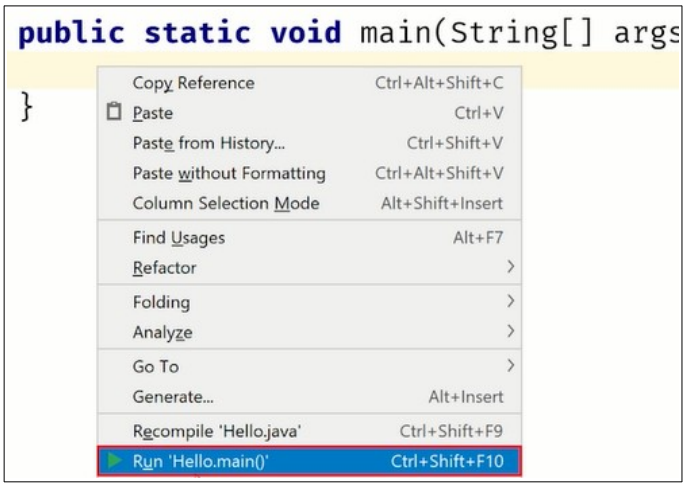
The left and right parenthesis in a method declaration are mandatory and can optionally include one or more parameters - which is a way to pass information to a method. More

To Run

1.



2.



Zero is used to indicate there was not any problem.

```
Process finished with exit code 0
```

Code block - A code block is used to define a block of code. It's mandatory to have one in a method declaration and it's here where we will be adding statements to perform certain tasks.

To print

```
System.out.print("My name is kazim"+x);
```

Variables

Variables are a way to store information in our computer. Variables that we define in a program can be accessed by a name we give them, and the computer does the hard work of figuring out where they get stored in the computers random access memory (RAM).

A variable, as the name suggests can be changed, in other words, it's contents are variable.

```
int x=3;  <!-- here the intialization is optional, we can omit the
                    value 3 →
```

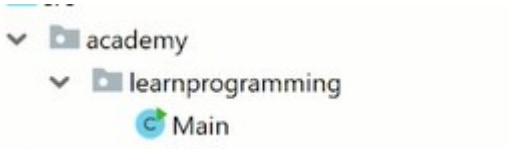
JAVA will compile and run the programme, read this statement we have created and allocate a place in memory to store a single whole number and will assing x to that area of memory.

The eight primitive data types in Java are **boolean**, **byte**, **char**, **short**, **int**, **long**, **float** and **double**. Consider these types as the building blocks of data manipulation. Let's explore the eight primitive types in Java.

Java Packages

A package is a way to organize your Java projects. For now, consider them as folders with **learnprogramming** in our example being a subfolder of **academy**. Companies use their domain names reversed.

So **learnprogramming.academy** becomes **academy.learnprogramming** - we will go into a lot more detail about packages later in the course.




```
package academy.learnprogramming;
```

```
int myMinIntValue = Integer.MIN_VALUE;
int myMaxIntValue = Integer.MAX_VALUE;
System.out.println("Integer Minimum Value = " + myMinIntValue);
System.out.println("Integer Maximum Value = " + myMaxIntValue);
System.out.println("Busted MAX value = " + (myMaxIntValue + 1));
```

```
Integer Minimum Value = -2147483648
Integer Maximum Value = 2147483647
Busted MAX value = -2147483648
```

This called Overflow incase of MAX_VALUE

in Case of MIN_VALUE is called underflow.

```
int myMaxIntTest = 2147483648;
```
Integer number too large

```
int myMaxIntTest = 2_147_483_647;
```

```
byte myMinByteValue = Byte.MIN_VALUE;
byte myMaxByteValue = Byte.MAX_VALUE;
```

```
Byte Minimum Value = -128
Byte Maximum Value = 127
```

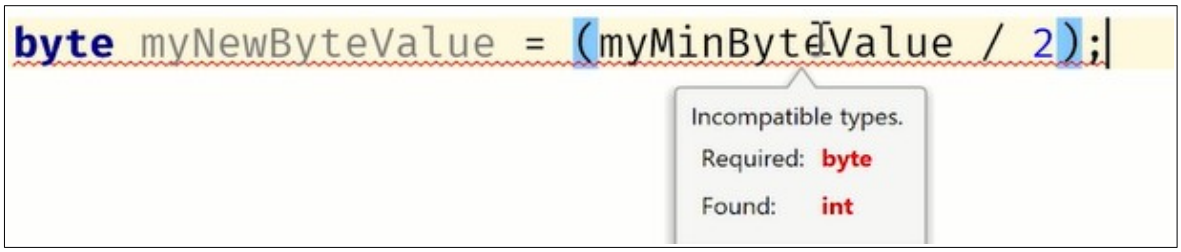
An **int**, has a much larger range as we know, and occupies **32 bits**, and has a width of **32**.

```
long myLongValue = 100L;
```
Long is twice big compared to 32.

```
long bigLongLiteralValue = 2_147_483_647;
long bigLongLiteralValue = 2_147_483_647_234;
```
Integer number too large

```
Short Minimum Value = -32768
Short Maximum Value = 32767
```

Casting in java



The above expression result into a number that fits into a byte. We know that, how we tell java that, we do by using the concept of casting.

```
byte mybyte= (byte) (z/2);
```

Float and double

```
int myIntValue = 5;
float myFloatValue = 5f;
double myDoubleValue = 5d;
```

Floating point precision

```
MyIntValue= 1
MyFloatValue= 1.6666666
MyDoubleValue= 1.6666666666666667
```

double has a better decision

This also works, but better practice is using "d"

```
double myDoubleValue = 5.00 / 3.00;
```

Double takes more space, modern computers deals with double faster than float, Java lib in particular MATH function are often written or return to process doubles and not floats.

Char and boolean Primitive data types:

```
char myChar = 'D';
```

for e.g to store the last key pressed by user in a game.

Char occupies 2 bytes. The reason its not just single byte is that it allows you to store unicode characters.

```
char myCopyrightChar = '\u00A9';
boolean myFalseBooleanValue = false;
```

Copyright symbol.

String is not a datatype its a class.

When I said you can delete characters out of a **String**, that's not strictly true. Because **Strings** in Java are immutable. That means you can't change a **String** after it's created. Instead, what happens is a new **String** is created.

So if u append something to a String variable, for e.g
x=x+"kazim";
Variable x doesnt get appended the value "kazim" instead a new String is created. With variable x value appended to kazim.

As a result of a **String** being created, appending values like this is inefficient and not recommended. I'll show you a better way of doing it in a future video where we discuss something called a **StringBuffer** which can be changed.

Operators, Operands and expressions.

An operand is a term used to describe any object that is manipulated by an operator.
e.g
int myVar= 15+12;
+ is the operator.
15 and 12 are the operands, var used instead of literals are also operands.

Comments: // , /* */

if-then
e.g1:

```
if (isAlien == true);
    System.out.println("It is not an alien!");
```

e.g2:

```
if(false)
    System.out.print("hello");System.out.print("bye");
```

the code block if not given is only for the first statement.

Logical AND-Operator.
== equal
!= not equal
>=, <=
&&
Logical OR-Operator.
|| -or operator.

AND , OR operator comes in two flavours;

&&,|| is logical and operators which operates on boolean operands.
&: bitwise operator woking at bit level.
| is a bitwise operator working at bit level.

Ternary operators:
we have 3 operands.

```
boolean wasCar = isCar ? true : false;
```

The first operand is a condition, which will evaluate to true or false.
If the first operand aka condition is true, its gonna **return true** , else its gonna **return false**.

So a ternary operator is a shortcut to assigning one of two values to a variable depending on a given condition.

```
boolean isEighteenOrOver = (ageOfClient == 20) ? true : false;
```

Summary of operators Summary of operators

Section 4:

– Any thing blue means its a reserved word (keyword)
– statements dont need be on one single line

```
System.out.print("hello"
            +"my"
            + "is kazim");
```

variables defined inside of a code block, are defined only inside the codeblock.

```
if(true){};
else{System.out.print('hello');}
```

'else' without 'if'

```
if(true){}
else if(true){
    System.out.print("hello");
}
else{System.out.print("hello");}
```

Methods in java
--pvsm , here the 'm' is method and the method is all the statements in it.

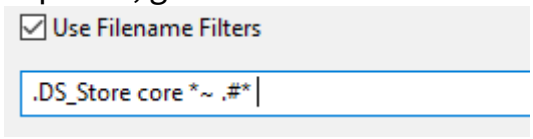
```
public static void myname(){

}
public static void myname(){

}
```

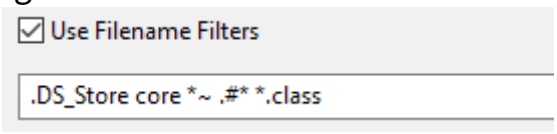
The grey color tell this method is never used

it turns to black when used

Diff merge tool

Diff merge is an program that will help u to visually compare and merge files on any operating system.
In options, goto folders filters and change this



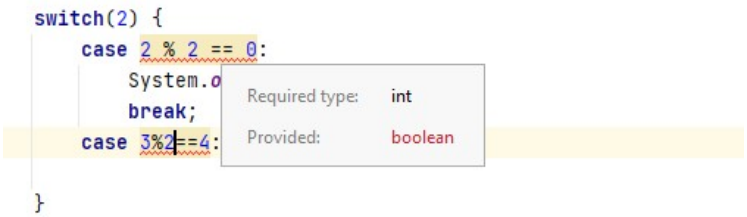


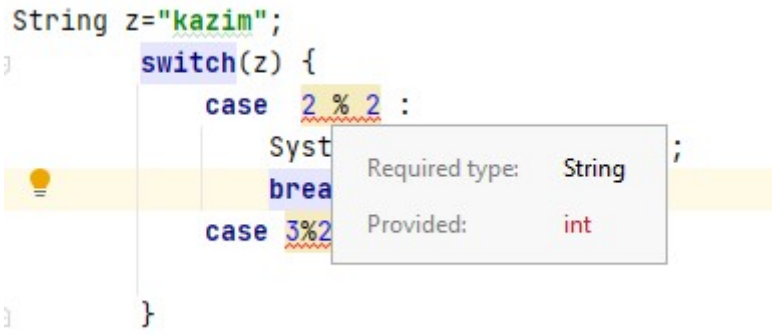
This will skip class files in filtration.

Method overloading creating methods with a used method name with different parameters datatype.

Section 5

Switch statement:
1) reason it simplifies if else if then <!-- used when two many else if statements →





Switch is not as good as **if**, as shown above.
Switch is good for if we testing on some variable and you wanna test different value for that variable.

```
case (7): case 3: case 4:
    break;
default:
    break;
```

** String class has methods.

For loop

```
for(int i=4; i<8;i++){
    System.out.print("hello");
}
```

While and do-while

-break statement breaks the loop and **continue statement** tells java to go back to the start of the loop instead.

Big difference between **while** and **for** loop Regarding **continue**

```
for(int i=4; i<8;i++){
    if(i==4){
        continue;
    }
    System.out.print(i);
}
```

This would lead to infinite loop in **while loop**

-- new Trick

to make sure this is a individual seperate block

to make sure the **int I** Is local to the block only

```
;{
    int i = 4;
    while(i<8){
        if (i==6)
            continue;
    }
    i++;
}
```

this would lead to infinite loop, no way of making this proper. If we put in the beforeBegin in while loop the increment will be from the next iteration. So u have to modify the conditional a bit to fit **for loop**
This is a poor desing but.

Opps made an error;

```
;{
    int i = 4;
    while (i < 8) {
        if (i == 6) {
            continue;
        }
        i++;
    }
}
```

Do while: statement make sure the code run atleast one time, and from the next it will run on the basis of conditional statement.

```
do {
    System.out.print("hello");
x++;
}while (x<5);
```

Parsing values from a String: That is to convert a String into an Int .

```
int num= Integer.parseInt( s: "1");
```

We are using a class Integer (Camelcase), this is a wrapper class for the primitive type int.
Wrapper class includes some useful static methods, the one we using is parseInt.
The method parseInt will try to convert argument into an integer.

{To convert the datatype we need to include the type we converting to, so that we able to get the parsing method associated with that type.}

```
int num= Integer.parseInt( s: "1");
```

ParseInt is a static method to convert String into an number.

```
Double.parseDouble( s: "1");
```

Reading user Input

```
Scanner scanner= new Scanner(System.in);
System.out.print("Enter ur name:");
String x= scanner.nextLine();
System.out.print("Hello "+x);
scanner.close();
```

System.out	<!-- dumbs texts on the console →
System.in	<!-- allows u type input into the console →
new	<!-- create a new instance or new object →

Note:
1.after using the scanner we should close the scanner using the close method.
2. after closing the scanner statement like nextLine or nextInt will cause errors, so we should really ensure we're done with using the scanner before closing it.
3. By closing the scanner, we release the underlying memory that scanner was using.

Scanner issue:

int age= scanner.nextInt() <!-- it will take the output as u imagine →
scanner.nextLine() <!-- To handle the enter key issue -->
String name=scanner.nextLine() <!-- It will now work as u imagine -->

Scanner.hasNextInt(): this will see to check if the next input entered is a integer, however it will not remove it from the scanner.

So iow , it will ask the user for input and check to see if it qualifies as an int, the method would return false. So it allows from generating type errors when using scanner.nextInt().

```
Scanner scanner= new Scanner(System.in);
boolean z=scanner.hasNextInt();
System.out.print(z);
```

```
Hello
"C:\Program Files\Amazon Corretto\jdk15.0.2_7\bin\java.exe
kazim
false
```

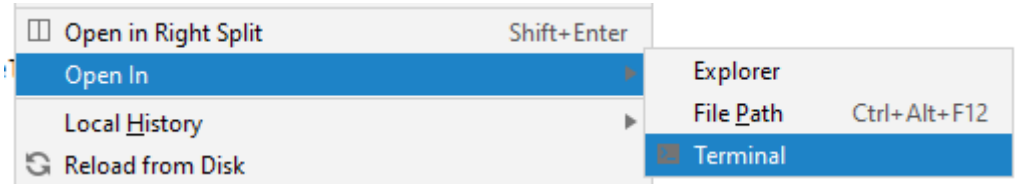
```
Scanner scanner= new Scanner(System.in);
boolean z=scanner.hasNextInt();
System.out.println(scanner.nextLine());
System.out.print(z);
```

It assign the value to the next scanner nextline.

```
Hello
"C:\Program Files\Amazon Corretto\jdk15.0.2_7\bin\java.exe" "-java
kazim
kazim
false
```

OOP 1. Classes, Constructor, inheritance

Object: real world object, state, behaviour.
Right click on folder src

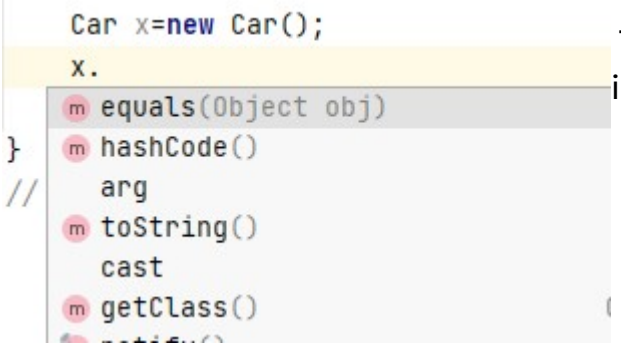


public is an access modifier that we used to determine what access that u want others to have to this new class that we creating. The word public means unrestricted access to the class.
Private: no other class can access that class.
Protected: which allow classess in this package to access this class.

Classes allow us to have variables that can be seen and our accessible within the class.
Class variable, member variable, fields.

General rule:
when u defining fields in java, use access modifier 'private' . Here we doing is encapsulation which is the key fundamental rule in OOP.
Which hide methods and fields from access publicly.
That means the internal representation of the object will be hidden from view, outside of object definition.
Again, the object will be created from the template of our class. We are allowing the internals to access the object, we are not allowing the outside to have access.

```
Car x=new Car();
x.
```



This are the methods, that the car class has inherited from the base java class.

```
Car x=new Car();
x.mynickname="Sahil";
```

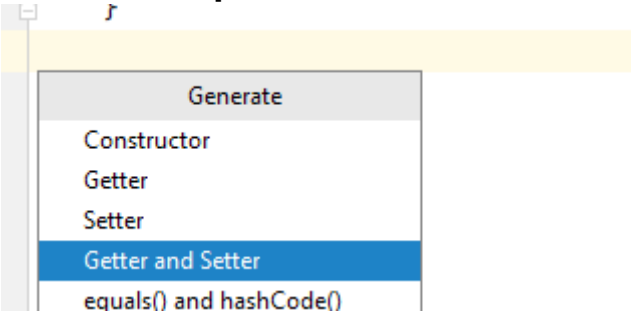
Cuz of public.
Bad way to set field value

```
public void setModel(String model){
    this.model=model;
}
```

Right way to set field value.

We can do validation on setter and getter method.
For e.g within a range, checking the type first so we dont get errors, within the set.

Constructors p1:



Instead of using multiple setter you can use a constructor

```
Car x=new Car();
x.setModel("Audi");
x.setMynickname("sahil");
x.setPrice(100000);
```

```
Car x=new Car();
```

When you doing this, u r actually calling the constructor.

```
public Car(){
    System.out.print(" Empty constructor called");
}
```

Note: You can call one constructor from another constructor.
For e.g: if an empty constructor is called, we wanna create an object with some default value.

```
public Car(){
    System.out.println(" Empty constructor called ");
    this( price: 1000, model: "Bugatti veryon", mynickname: "sahil");
}
public Car(String model,in
```

Call to 'this()' must be first statement in constructor body

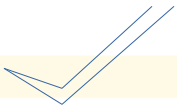
>>>

```
Constructor called
Empty constructor called
sahil
```

```java
public Car(String model,int price, String mynickname){
    System.out.println("Constructor called");
    this.model=model;
    this.price=price;
    this.mynickname=mynickname;

}
```

Partially correct? cuz

```java
public Car(int price,String model, String mynickname){
    System.out.println("Constructor called");
    setPrice(price);
    setModel(model);
    setMynickname(mynickname);

}
```



The Advantage if there is a validation, that code would be executed too.

This two are conflicting in which is better, in java when talking about subclassing this using setter and getter wont always work. General rule of thumb is to save the value for the constructor directly into the field cuz there would be scenarios that the code in the setter isnt executed.
Also dont use setter and getter, cuz some aspect of initialization maynot be finished, while u r in the constructor.
So dont use setter and getter in the constructor.

```java
public Car(String color,String seater, int price){
    this( model: "ferrari", price,  mynickname: "mylove",color,seater);
}
 Car x= new Car( color: "red", seater: "four", price: 100000);
 System.out.print(x.getMynickname());
```

Inheritance

Different kinds of an object have certain amount of field in common with each other.

```java
public class Dog extends Animal{

}
```

There is no default constructor available in 'Animal'

Create constructor matching super Alt+Shift+Enter More

```java
public Dog(String name, String brain, String wieght, String size, String body) {
    super(name, brain, wieght, size, body);
}
```

what super means call the constructor from the class we are extending from or inheriting from.
So we initialize the animal class, n dog is a part of an animal class.
Now, we can add more features to the dog class, things that are unique to a dog and not thus a very generic animal.

In the animal class we can add extra field and methods that are unique for a dog and not necessarily for an animal.

```
@Override
public void eat() {
    super.eat();
    System.out.print("dog is eating");
}

public void run(){
    move();
    System.out.print("Dog is running");
}
```

super.move() <!-- super keyword is optional. If u didnt use super keyword, it may get over-ride by a Override method in dog class.

Reference vs object vs instance vs class

A class is basically a blueprint for a house, using the blueprint plans we can build as many houses as we like based on those plans.
Each house you build(in other words **instantiate** using the **new** operator) is an object also known as instance.
Each house you build has an address(a physical location). IOW if u want to tell someone where you live, you give them your address (perhaps written on a piece of paper). This is known as reference.
You can copy that reference as many times as you like but there is still just one house.
IOW we are copying the paper that has address on it not the house itself.
We can pass references as parameters to constructor and methods.

Link: r[eference vs object vs instance vs class]

```java
public class House {
    private String color;

    public House(String color) {
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }
}
```

```java
public static void main(String[] args) {
    House blueHouse= new House( color: "blue");
    House anotherHouse=blueHouse;

    System.out.println(blueHouse.getColor());     //print blue
    System.out.println(anotherHouse.getColor());  // blue

    anotherHouse.setColor("red");
    System.out.println(blueHouse.getColor());     //print red
    System.out.println(anotherHouse.getColor());  // red

    House greenHouse = new House( color: "green");
    anotherHouse=greenHouse;

    System.out.println(blueHouse.getColor()); //red
    System.out.println(greenHouse.getColor()); //green
    System.out.println(anotherHouse.getColor()); //green

    // main ends here
}
```

Here we have a class House with an instance variable (field) color. On the right hand side we have the Main class with the main method. This code is creating instance of the House Class, changing the color and printing out the result.

This vs super

The keyword **super** is used to access/ call the parent class members (variables and methods).

The keyword **this** is used to call the current class members(variables and methods). This is required when we have a parameter with the same name as a instance variable (field).

NOTE: We can use both of them anywhere in a class except static areas(the static block or a static method). Any attempt to do so will lead to compile-time errors.

The keyword super is used for method overriding.

() --→ call
Use this() to call a constructor from another overloaded constructor in the same class.

For later

The java compiler puts a default call to super() if we dont add it, and it is always the no args super which is inserted by compiler(constructor without arguments);

Even Abstract classes have constructors, although you can ne ver instantiate an abstract class using the new keyword.

An abstract class is still a super class, so its constructor run when someone makes an instance of concrete subclass.

A constructor can have call to super() or this() but never both.

Another e.g

```
public class Rectangle {
    private int x;
    private int y;
    private  int width;
    private int height;

    public Rectangle(){
        this( width: 0, height: 0);
    }
    public Rectangle(int width, int height){
        this( x: 0, y: 0,width,height);
    }
    public Rectangle(int x, int y, int width, int height){
        this.x=x;
        this.y=y;
        this.width=width;
        this.height=height;
    }

}
```

⚠ 1

The first constructor calls the 2nd, the 2nd calls the third, 3rd constructor initializes the instance variables.

The 3rd does all the work.

This is known as constructor chaining. The last constructor responsibility to initialize the variables.

Another e.g

```
class Shape{
    private  int x;
    private  int y;

    public Shape(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
public class Rectangle extends Shape {

    private  int width;
    private int height;


    public Rectangle(int x, int y){
        this(x,y, width: 0, height: 0);
    }

    public Rectangle(int x, int y,int width, int height) {
        super(x, y);
        this.height=height;
        this.width=width;
    }
}
```

Method Overriding and Method overloading

Method Overloading:
means providing two or more seperate methods in a class with the same name but different parameters
method return type may or may not be different and that allows us to reuse the same method name.
Overloading is very handy, it reduces duplicated code and we dont have to remember multiple method names.
Overloading does not have anything to do with polymorphism but java developers often refer to overloading as Compile Time polymorphism.
In other words the compiler decide which method is going to be called based on the method name, return type and argument list.

We can overload static and instance methods.

Usually overloading happens inside a single class, but a method can also be treated as overloaded in the subclass of that class.

This is because the subclass inherits one version of the method from the parent class and then the subclass can have another overloaded version of the method.

Method Overloading Rules

Method will be considered overloaded if both follow the following rules

- Methods must have the same method name.
- Methods must have different parameters.

If methods follows the rules above then they may or may not

- Have different return types
- have different access modifiers.
- Throw different checked or unchecked exceptions.

Method Overriding

defining a method in a child class that already exist in the parent class with the same signature(same name, same arguments).
By extending the parent class the child class gets all the methods defined in the parent class(those methods are also known as derived methods).
Method overriding is also known as Runtime polymorphism and Dynamic Method dispatch , because the method that is going to be called is decided at rutime by JVM.

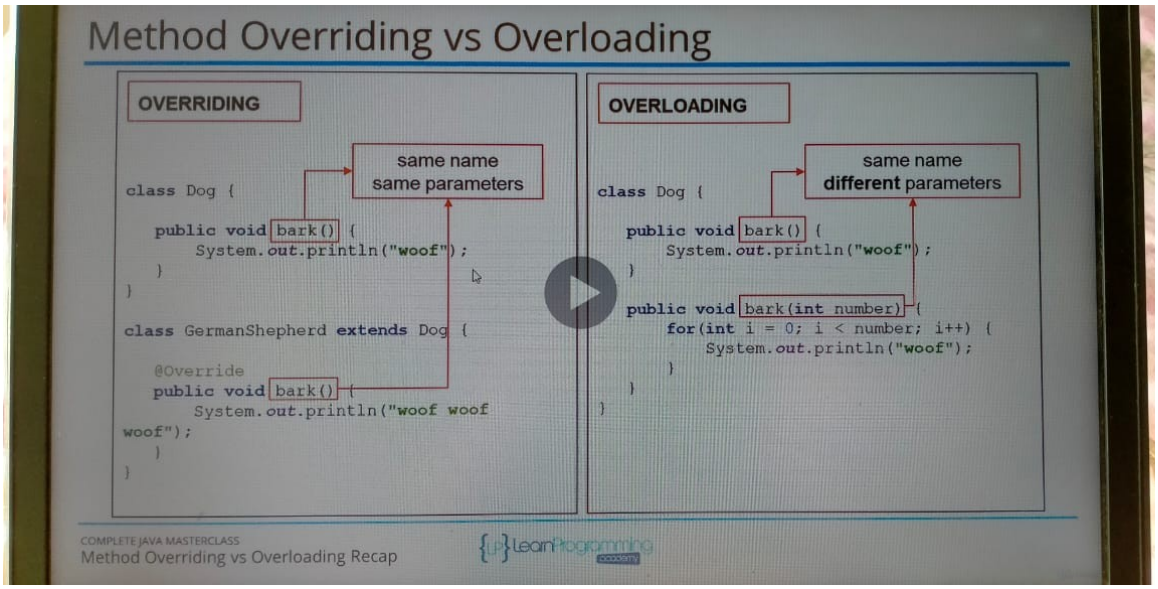
When we override a method its recommend to put @Override immediatetely above the method definition. This is an annotation that the compiler reads and will show us an error if we dont follow overriding rules correctly.

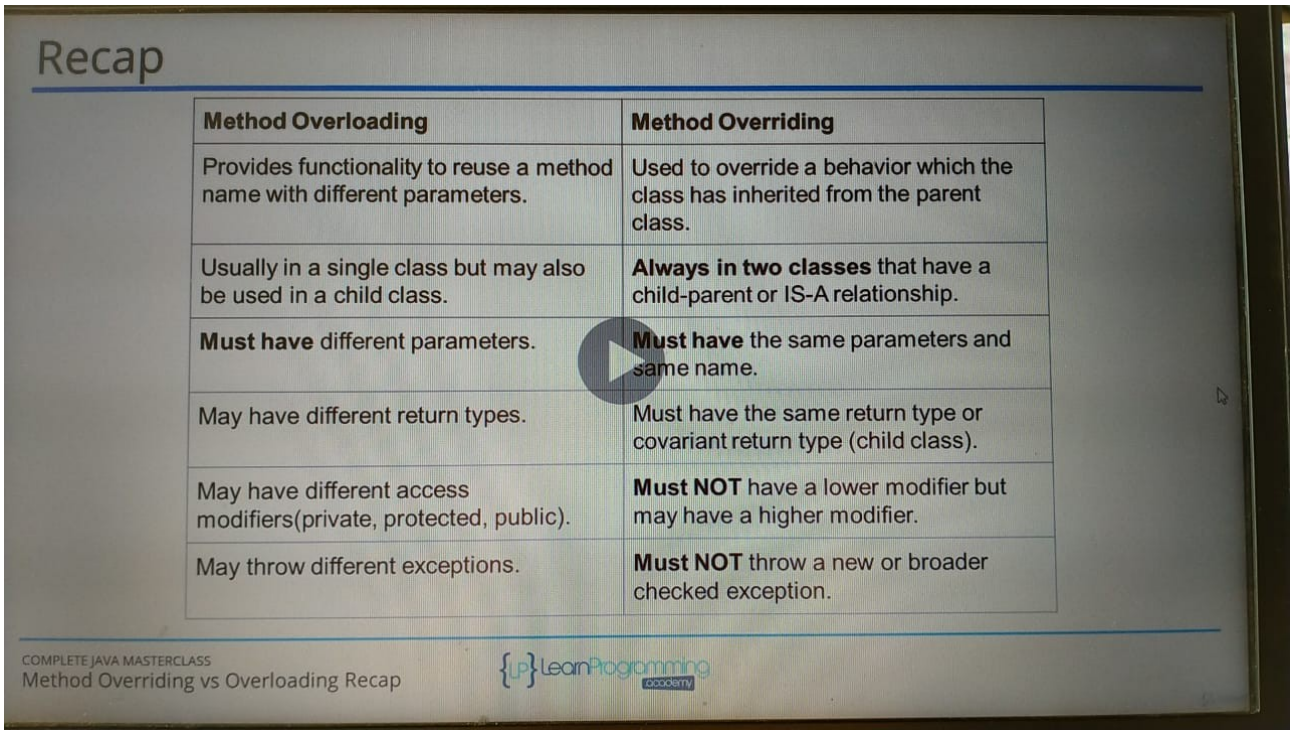
We cant over-ride static methods only instance methods.

Methods will be considered overriden if we follow these rules

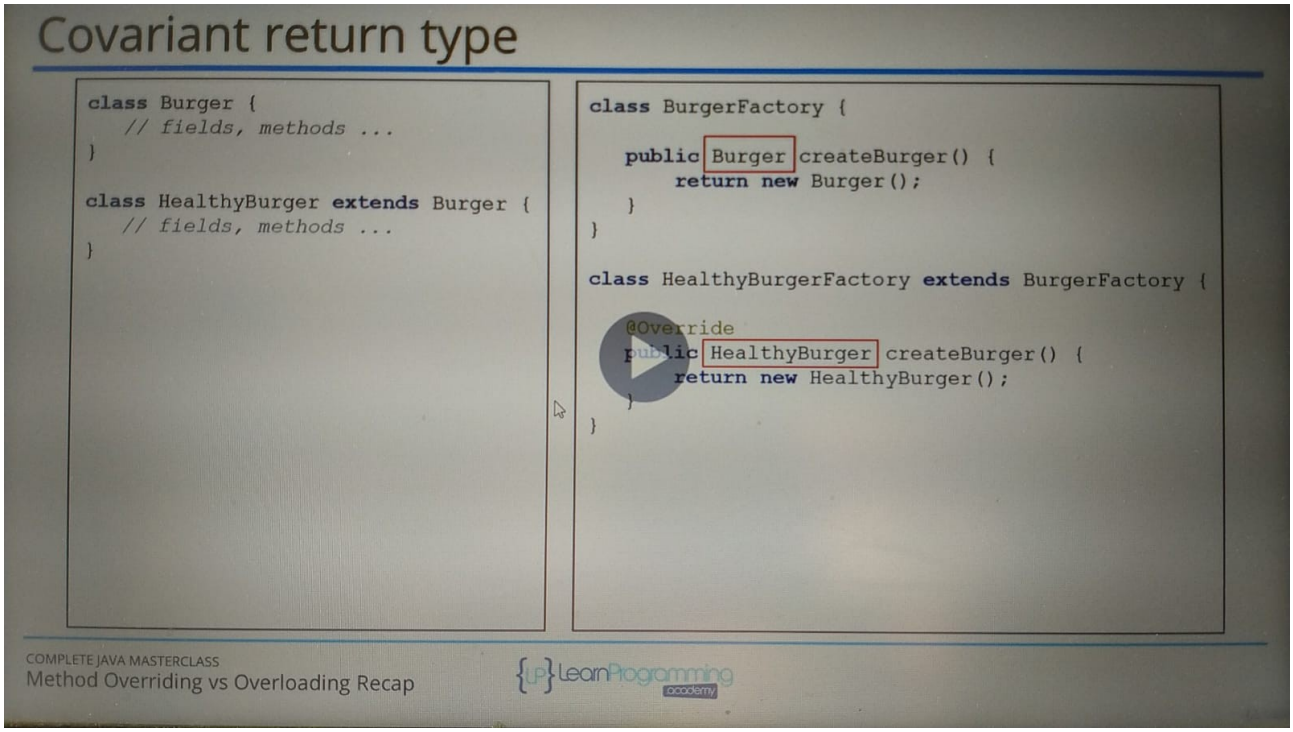
- It must have same name and same arguments
- Return type can be a subclass of the return in the parent class.
- It cant have a lower access modifer.
- For e.g if the parent method is protected then using private in the child is not allowed but using the public in the child would be allowed.

Important point about method overriding
only inherited methods can be overridden
constructors and private methods cant be overriden
Methods that are final cant be overriden.
A subclass can use super.methodName to call the superclass version of an overriden method.





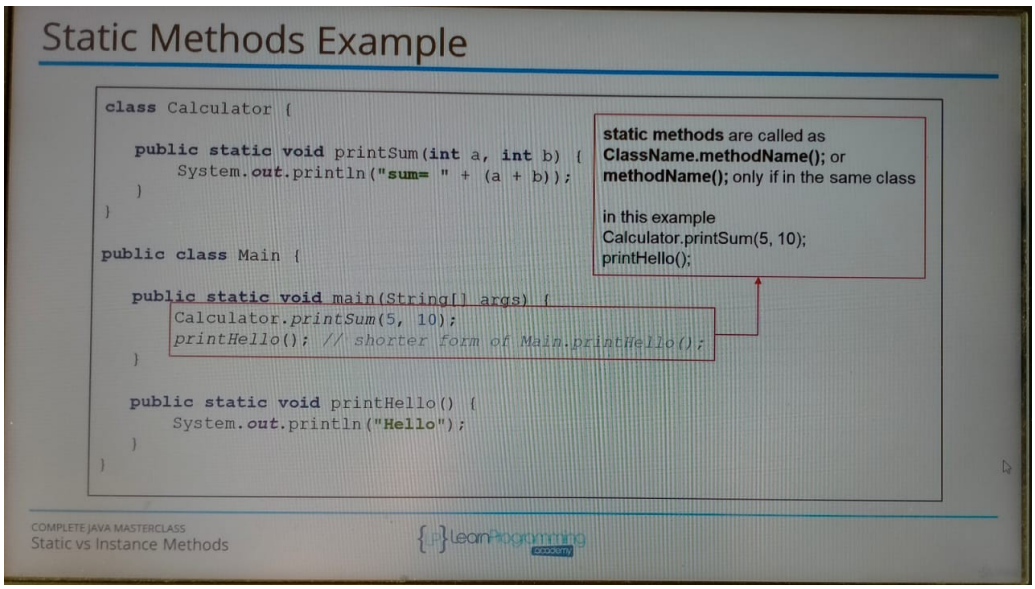
covariant type:



Static vs Instance methods

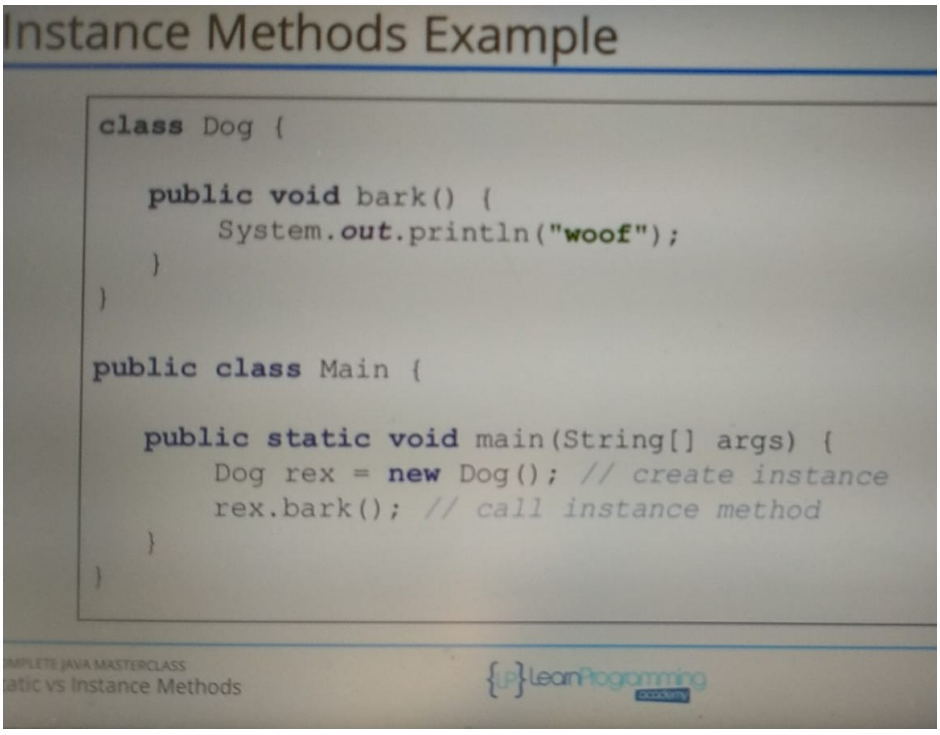
Static methods ared declared using a static modifier
Static methods cant access instance methods and instance variables directly.
They are usually used for operations that dont require any data from an instance of the class(from ‘this’)
if you remember, the **this** keyword is the current instance of a class.

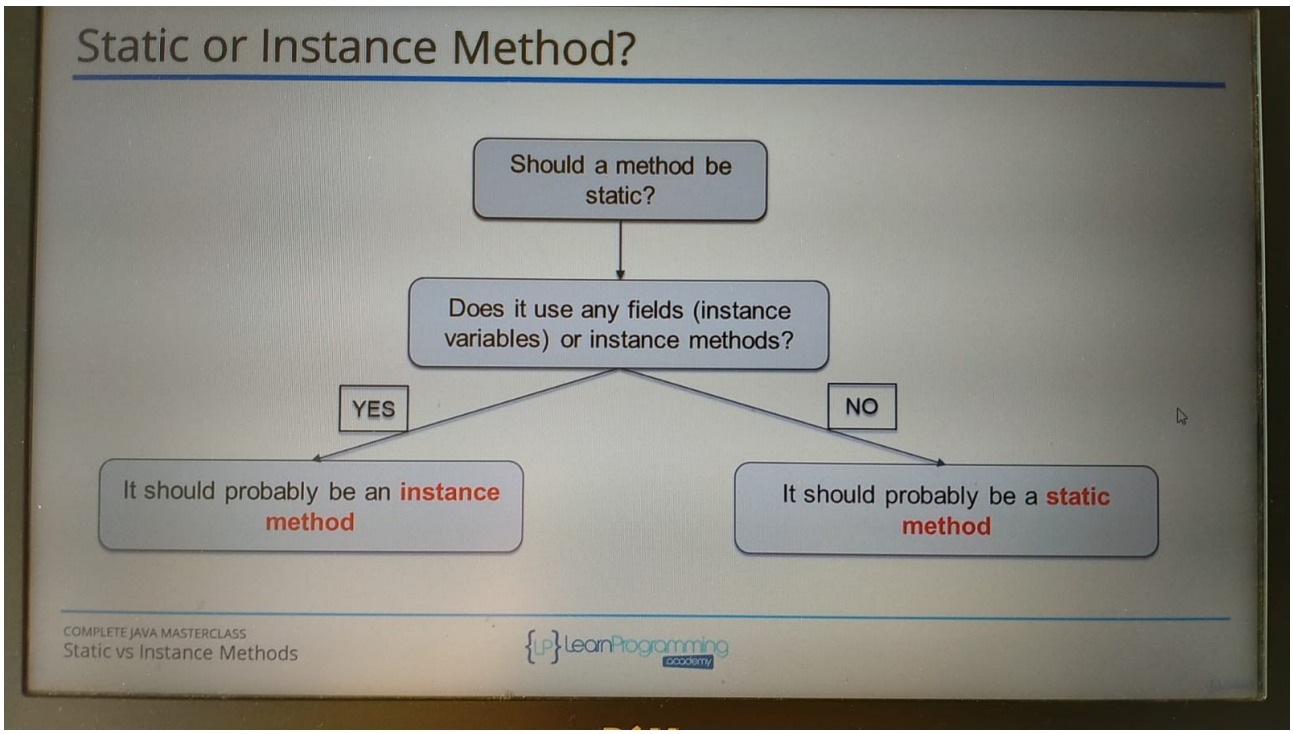
In static methods we cant use **this** keyword.
Whenever you see a method that does not use instance variables that methods should be declared as a static method.
For example main is a static method and its called by the JVM when it starts an application.



Instance Methods

Instance methods belong to an instance of a class.
To use instance method we have to instantiate the class first usually by using the new keyword.
Instance methods can access instance methods and instance variables directly.
Instance methods can also access static methods and static variables directly.

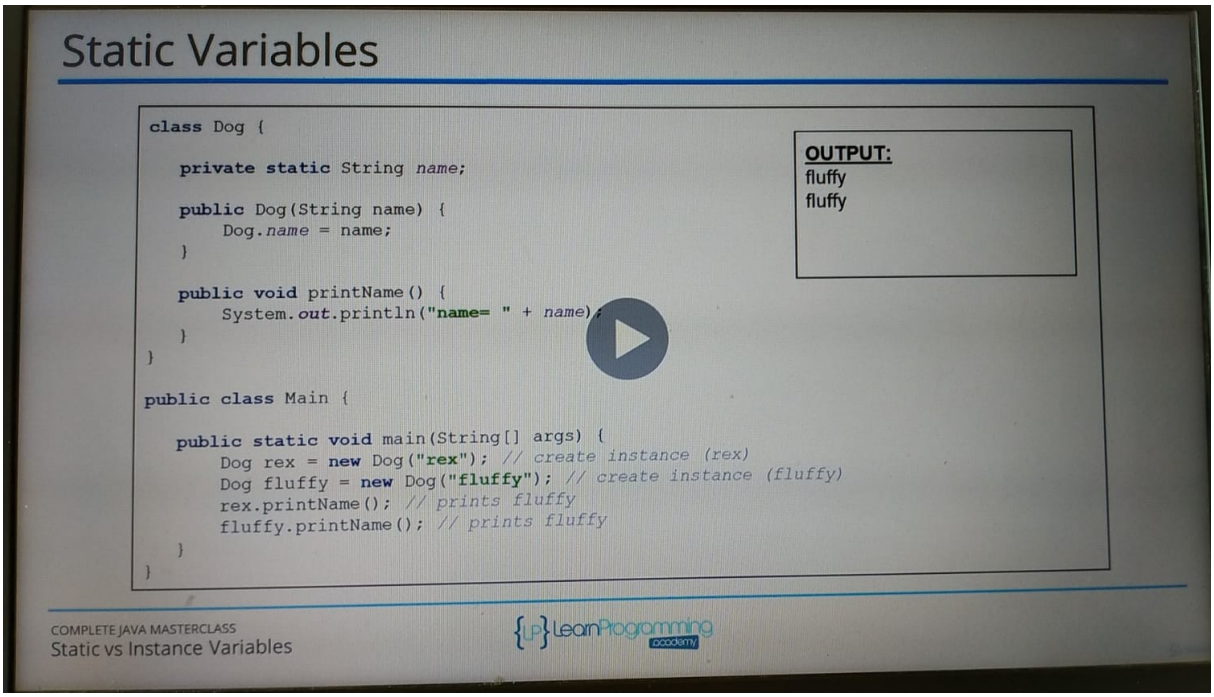




Static vs instance variables

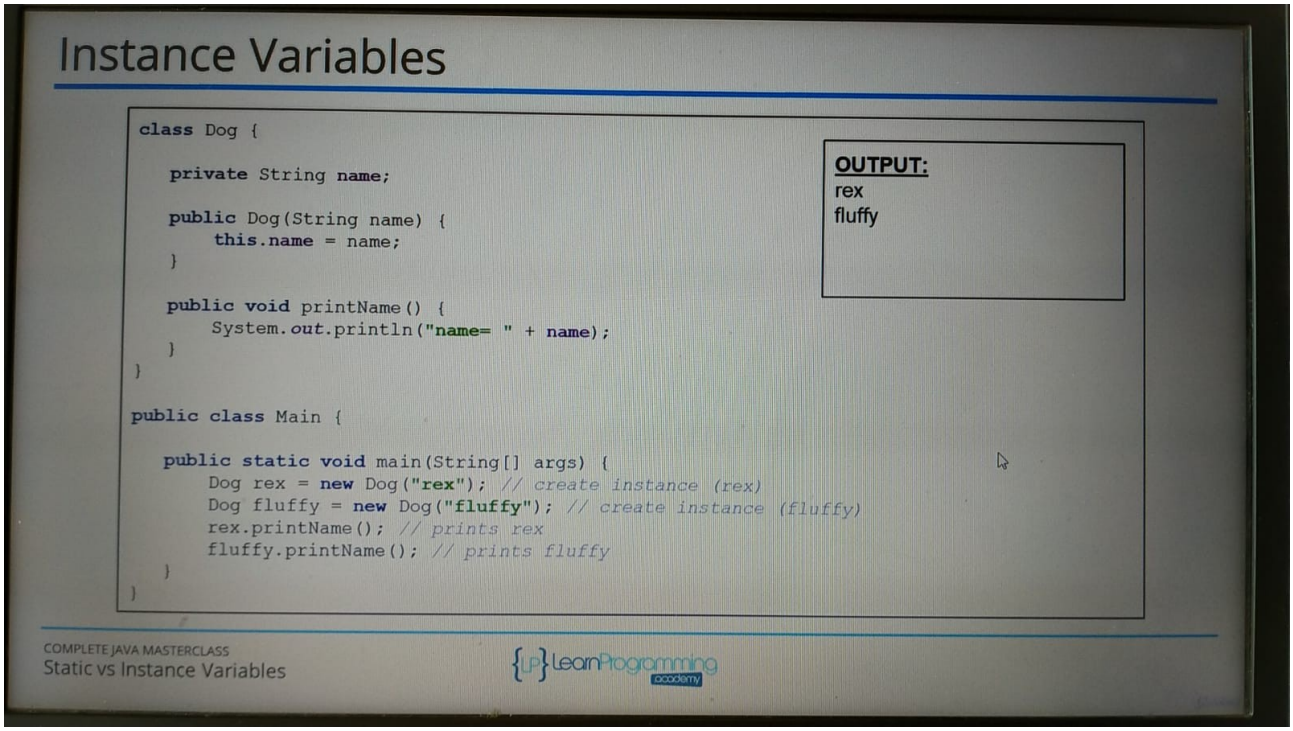
Declared by using the keyword static.
Static variables are also known as static member variables.
Every instance of that class shares the same static variable.
If a changes are made to that variable, all other instances will see the effect of the change.

Static variables are not used very often but can sometimes be very useful.
For e.g when reading user input using Scanner we will declare Scanner as a static variable.
That way static methods can access it directly.



Instance variables

They dont use the static keyword.
Instance variables are also known as fields or member variables.
Instance variables belong to an instance of a class.
Every instance has its own copy of an instance variable.
Every instance can have a different value(state).
Instance variables represent the state of an instance.



OOP part 2. Compostion, Encapsulation and Polymorphism

Inheritance deals with IS-A relationship : means a car IS-A vechile.
Composition deals with something different thats HAS-A relationship.
Consider a computer, a computer has a motherboard,case, monitor.
Motherboard , case , monitor are not computers in the same sense a car is a vechile.
But a computer has a motherboard, a computer has a case , a computer has a monitor.

Thats what composition is, its actually modeling the parts of a greater whole.

Inheritance:

```
public class Car1 extends Vechile{

    private int doors;
    private int engineCapacity;

    public Car1(String name, int doors, int engineCapacity
        super(name);
        this.doors=doors;
        this.engineCapacity=engineCapacity;
    }


}
```

Composition:

```
public class Monitor {                                          ⚠ 11
    private String model;
    private  String manufacturer;
    private int size;
    private Resolution nativeResolution;

    public Monitor(String model, String manufacturer, int size,
                   Resolution nativeResolution) {
        this.model = model;
        this.manufacturer = manufacturer;
        this.size = size;
        this.nativeResolution = nativeResolution;
    }
    // constructor ends
```

Monitor isnt a resolution.

Monitor has a resolution.

Another e.g

```
public class Case {                                                    ⚠ 11
    private String model;
    private String manufacturer;
    private String powerSupply;
    private Dimensions dimensions;

    public Case(String model, String manufacturer, String powerSuppl
        this.model = model;
        this.manufacturer = manufacturer;
        this.powerSupply = powerSupply;
        this.dimensions = dimensions;
    }
```

A Case has a Dimension.

```
public class PC {                                                      ⚠ 8
    private Case theCase;
    private Monitor monitor;
    private Motherboard motherboard;

    public PC(Case theCase, Monitor monitor, Motherboard motherboard)
        this.theCase = theCase;
        this.monitor = monitor;
        this.motherboard = motherboard;
    }
```

Pc has a case, pc has a monitor, pc has a motherboard.

```
Dimensions dimensions=new Dimensions( width: 20, height: 20, depth: 5);
Case theCase = new Case( model: "2208", manufacturer: "Dell", powerSupply: "240",dimensions);
Resolution nativeResolution= new Resolution( width: 2540, height: 1440);
Monitor theMonitor = new Monitor( model: "27inch Beast", manufacturer: "Acer", size: 27,nativeResolution);
Motherboard themotherboard=new Motherboard( model: "bj-200", manufacturer: "asus", ramSlots: 4,, bios: "v2.44");

PC thePC = new PC(theCase,theMonitor,themotherboard);

thePC.getMonitor().drawPixelAt( x: 1500, y: 1200, color: "red");
thePC.getMotherboard().loadProgram( programName: "Windows 1.0");
thePC.getTheCase().pressPowerButton();
```

```java
public void powerUp(){
    theCase.pressPowerButton();
    drawlogo();
}

private void drawlogo(){
    // fancy graphics
    monitor.drawPixelAt( x: 1200, y: 50, color: "yellow");
}

private Case getTheCase() {
    return theCase;
}
```

Better way is to make getter and setter private and control the execution. And if u writing a method use directly the object.

General rule, when designing programs in java, u wanna look at composition first b4 inheritance.

Encapsulation

Encapsulation is a mechanism that allows u to restrict access to certain components in the objects u creating so u r able to control the members of the class from external access. In order to guard against unauthorized access. We are not talking about security here. We are stopping outsiders for e.g classes or code outside the class you working on from accessing the inner working of the class.

Its useful to hide the innerworking from the otherclass to give u more control and to be able to change things without breaking the code elsewhere.

A Class that doesnt use Encapsulation E.g: A player having health -1 by some damage value.

Not using Encapsulation

1) the class of which instance is created, we are able to change those fields directly.
2) If we decided to change some field name of a class, that change doesnt need to be reflected in the code. Even for getter and setters. So the change is internal, when using encapsulation.
3) Validation using constructor and without (by using getter and setter)

```java
public class Player {
    private String name;
    private int health=100;
    private String weapon;

    public Player(String name, int health, String weapon) {
        this.name = name;
        if(health>0 && health<=100){
            this.health=health;
        }
        this.weapon = weapon;
    }
}
```

Above if u user enter health not following range[1,100] it will set it as 100.
The other way is to use a method.

How to rename all like ctrl+H in intellij
select the field or any char's.
Right click then Goto refactor and rename; it will change it at runtime as u type.

Polymorphism

allow actions to act differently based on the actual object, the action is been performed on.

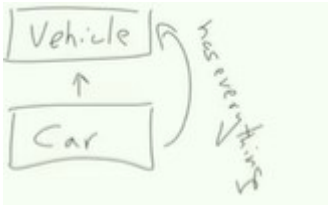
e.g
We gonna create classess in the same file as main, for having classes that arent going to be reused. The advantage of this, its really only useful to create classes in the same java source file if they are quite small and compact.

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

        for(int i=0; i<11;i++){
            Movie movie=randomMovie();
            System.out.println("Movie # "+i+":"+movie.getName()
            +"\n"+"Plot:"+ movie.plot() +"\n"
            );
        }

        // main ends here
    }
    public  static  Movie randomMovie() {
        int randomNumber = (int) ((Math.random() * 3) + 1);
        switch (randomNumber) {
            case 1:
                return new StrangerThings();
            case 2:
                return new Friends();
            case 3:
                return new GameOfThrones();
            case 4:
                return new Forgetable();

        }
        return null;
    }
}
class Movie{
    private  String name;

    public Movie(String name) {
        this.name = name;
    }
    public  String plot(){
        return "No plot here";
    }

    public String getName() {
        return name;
    }
}

class StrangerThings extends Movie{

    public StrangerThings() {
        super( name: "name");
    }
    @Override
    public  String plot(){
        return " Kid goes missing ";
    }

}
```

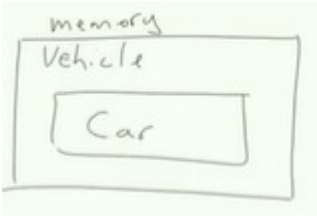
OUTPUT:

```
Movie # 0:Game of thrones
Plot:War of the 5 kings

Movie # 1:Friends
Plot:Friend stays together and say funny things
```

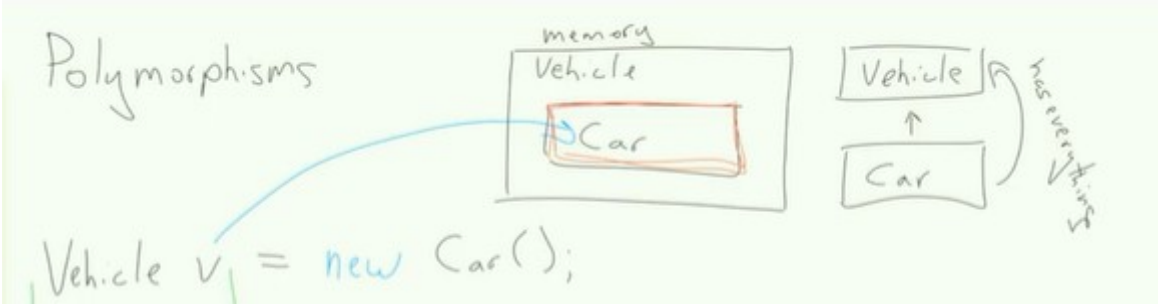
Lecture video : for [polymorphism]



which means when it creates the memory space for vechile it means we can put anything inside of this memory space that inherits from vehicle.
Bcoz we know that a car would be able to do anything that a vechile can do.



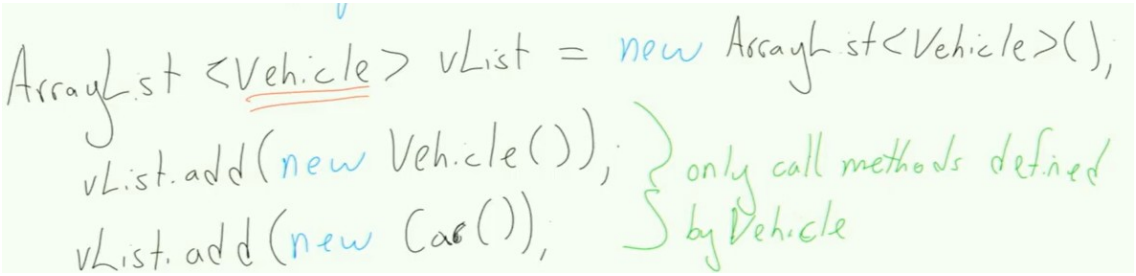
with object v I can only use vehicle methods, even though it holds the car objects.
Also if Car overriden any methods of Vechile, method of car will be call.



So I can use only Vechile methods, but when I call them it will use Car objects.

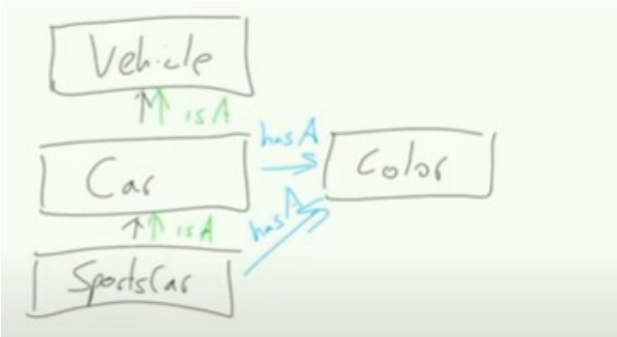
You can do this, btw

```
}
@Override
public  String plot(){
    getCheck1();
    return " Kid goes missing ";
}
public static void  getCheck1(){
    System.out.print("hello");
}
```



IS-A VS HAS-A

car is a vechile,
car has a color,



Polymorphism is we are assigning different functionality depending on the type of object we are creating.

Section 8. Arrays, Java inbuilt Lists, Autoboxing and Unboxing.

In arrays u can store multiple values of the same type.

Array is a datastructre that allows you to store values that all are of the same type.

e.g

```
int[] myIntArray=new int[10];
myIntArray={1,2,3,4};
```

cant be both

so just

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

e.g

```
int[] myIntArray=new int[10];
myIntArray[4]=45;
```

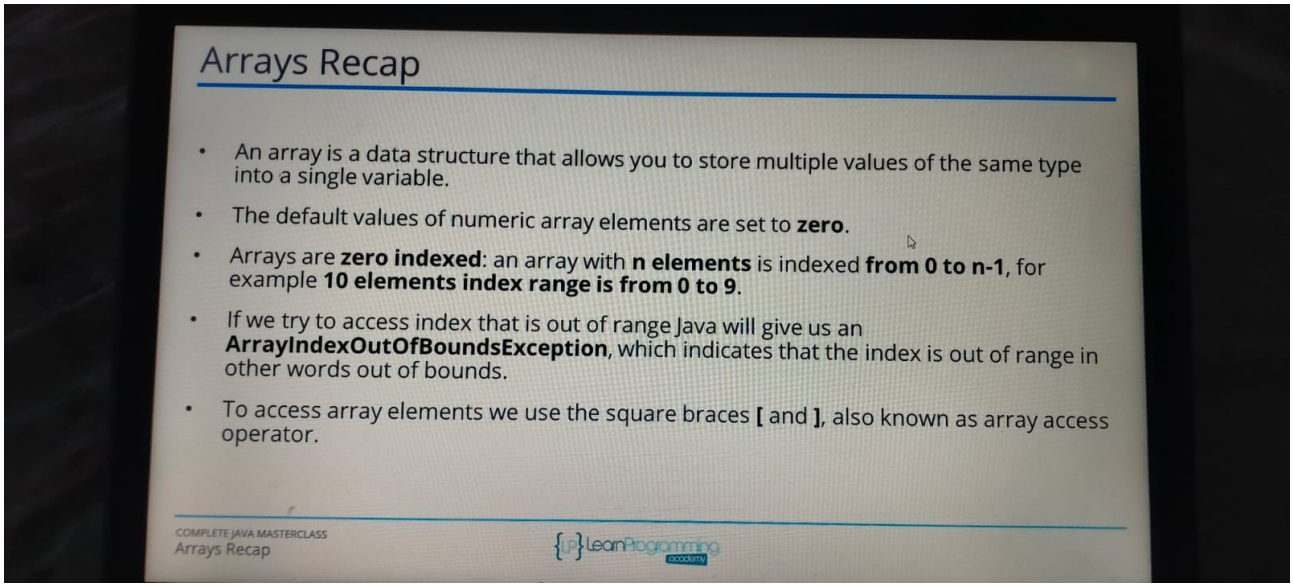
e.g

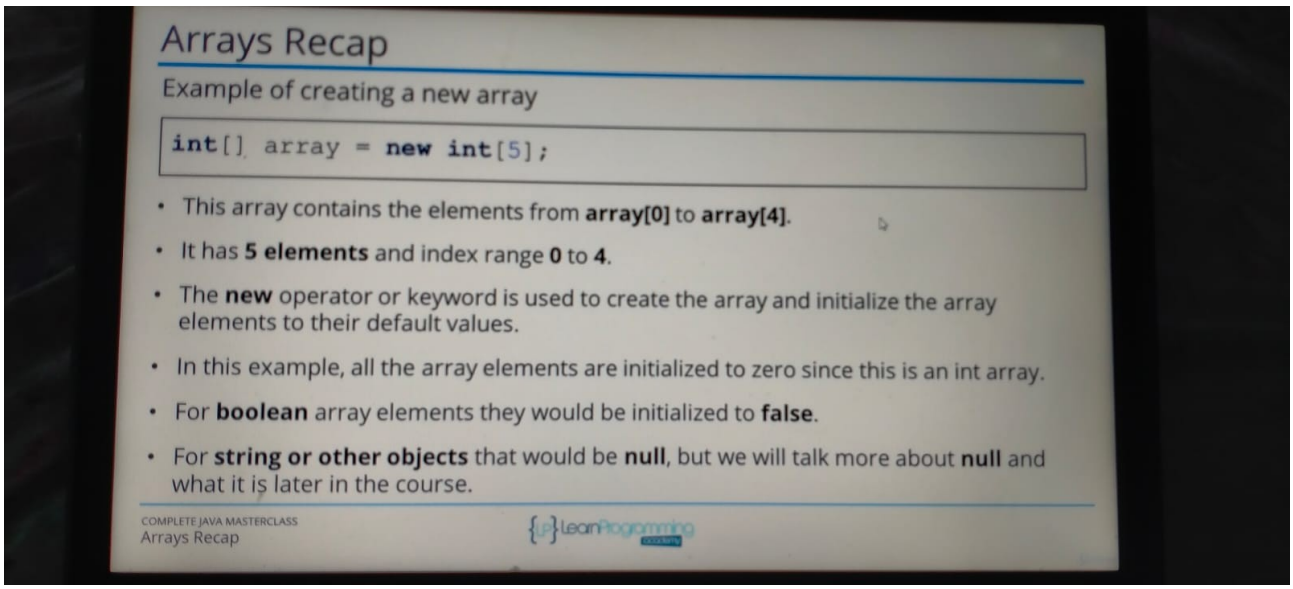
```
public static  void printArray(int[] array){

}
```

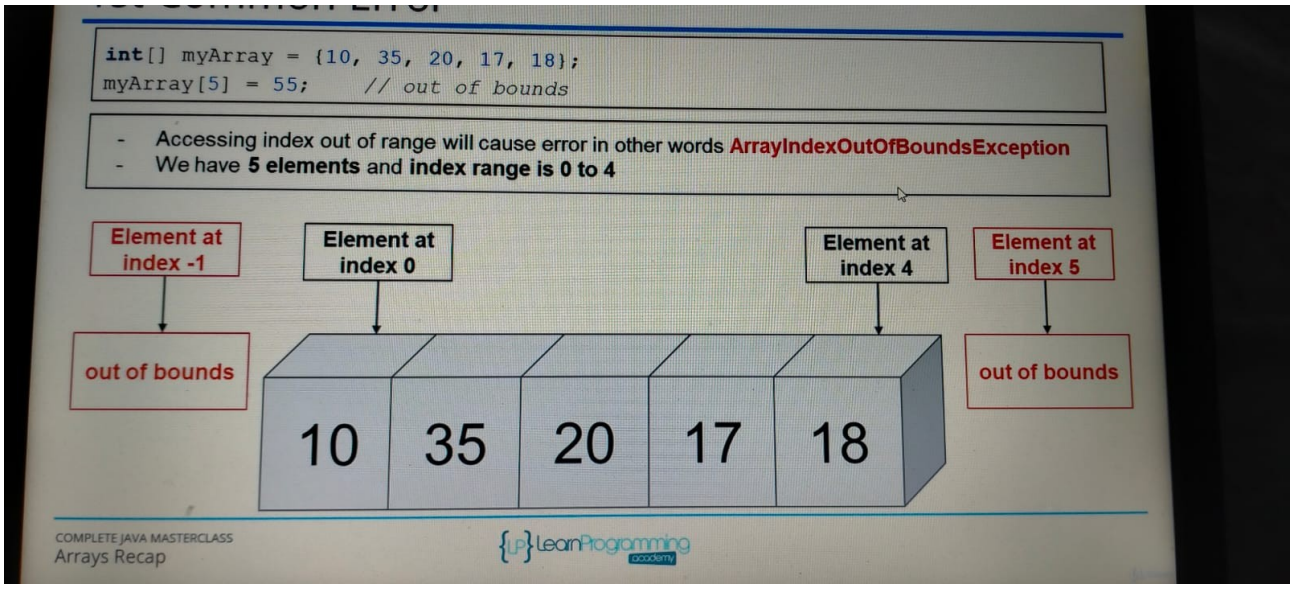
Java.util means thats something come with java .

Arrays recap

Array is a data structure that allows you to store multiple values of the same type into a single variable.







Reference type vs Value Type

```
int myIntValue=10;
int anotherIntValue=myIntValue;
System.out.println("myIntValue="+myIntValue);
System.out.println("anotherIntValue ="+ anotherIntValue);
anotherIntValue++;
System.out.println("myIntValue="+myIntValue);
System.out.print("anotherIntValue ="+ anotherIntValue);
```

```
myIntValue=10
anotherIntValue =10
myIntValue=10
anotherIntValue =11
```

Above thats value type.
So when we create an int variable(this value type), a single space in memory is allocated to store the value and that variable directly holds the value.

Now if u assign it to another variable, the value is copied directly. Then both variable work independently.

Reference type like array and classess works differently.
A reference type hold the address of the object but not the object.

```
int[] myIntArray= new int[5];
int[] anotherArray=myIntArray;
```

so, here myIntArray is actually a reference to the array in memory.
Here anotherArray is actually another reference to the same object in memory.

So myIntArray and anotherArray holds the same address in memory.
One way to know the reference type is by the **new** operator cuz it creates a new object in memory.
New Means new object.
On the second line we havent use the new keyword, we just use the equal sign.
To print the content of An Array

```
System.out.println("myIntArray= "+ Arrays.toString(myIntArray));
```

Prints the content of the entire array, this method “toString” joins multiple Strings and Objects into a String using a comma as a seperator.

```
myIntArray= [0, 0, 0, 0, 0]
anotherArray= [0, 0, 0, 0, 0]
```

```
anotherArray[0]=1;
System.out.println("myIntArray= "+ Arrays.toString(myIntArray));
System.out.println("anotherArray= "+ Arrays.toString(anotherArray));
```

```
myIntArray= [1, 0, 0, 0, 0]
anotherArray= [1, 0, 0, 0, 0]
```

```
if(showTransactions){
    System.out.print("\n"+customers.get(i).getName()+
            "--->"+customers.get(i).getTransaction().toString()

    );
}
```

another e.g

```
public static  void modifyArray(int[] array){
        array[0]=5;
}
```

```
modifyArray(myIntArray);
```

whats gonna happen when we pass the reference type to the method, the address is gonna be copied to the parameter

Another e.g

```
public static  void modifyArray(int[] array){
        array[0]=5;
        array= new int[] {1,2,3,4,5};
}
```

u have to use new int[] to initialize the new array, if the content is on the left and right curly braces. We are de-referncing the array.

You can also write like this

```
int[] myarray= new int[] {1,2,3,4,5};
```

List and ArrayList Part1

-Rezing is the problem with array, so we have List and ArrayList
-ArrayList is a re – sizable array which means it handles resizing automatically.
-List is just a interface

```
private ArrayList<String> groceryList = new ArrayList<String>();
```

we provide a constructor cuz ArrayList is a class, unlike

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

was just a primitive type.

– here we dont have any fixed size array number, see above.

Methods:

```
place2Visit.isEmpty();
```

```
groceryList.add(item);
```

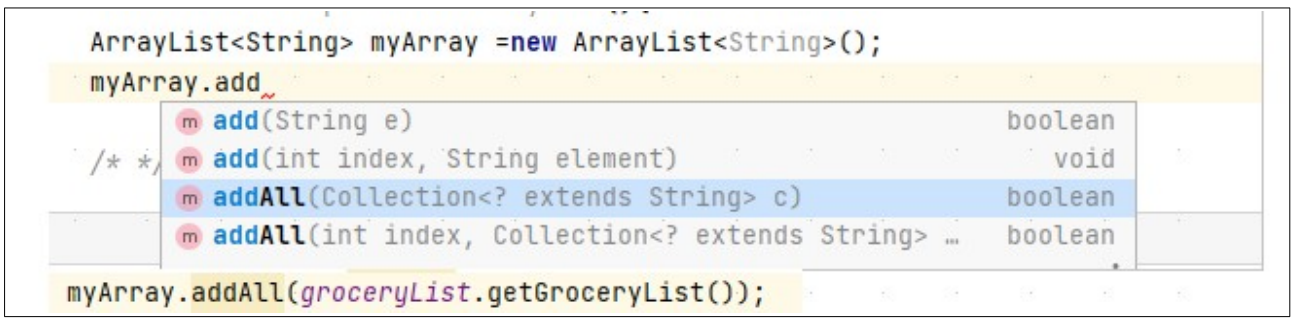
```
groceryList.set(position, newItem);
```

```
groceryList.remove(position);
```

```
groceryList.get(i)
```

```
boolean exist= groceryList.contains(searchItem);
```

```
int position = groceryList.indexOf(searchItem);//-1 Not found
```




```
ArrayList<String> myArray =new ArrayList<String>(groceryList.getGroceryList());
```

```
//shallow copy Lec 155 5:00
// Same address.
List<Theatre.Seat> seatCopy = new ArrayList<>(theatre.seats);
printList(seatCopy);

Collections.reverse(seatCopy);
Collections.shuffle(seatCopy);
System.out.println("Printing seatCopy");
printList(seatCopy);//prints reverse(desc)
System.out.println("Printing Theatre seats");
printList(theatre.seats);//prints (ascending)

Theatre.Seat minSeat = Collections.min(seatCopy);
Theatre.Seat maxSeat = Collections.min(seatCopy);
System.out.println("Min seat number is "+ minSeat.getSeatNumber());
System.out.println("Max seat number is "+ maxSeat.getSeatNumber());
```

```
return this.myContacts.indexOf(contact);// mycontacts is an array and Contact is a class
//of name and phone no.
// its gonna return an integer, telling what the element position is
//for that object in the array

String[] myArray =new String[groceryList.getGroceryList().size()];
 myArray = groceryList.getGroceryList().toArray(myArray);
// we pass myArray as argument to specify to what to update to
```

Array list part 3 time 16:00

and he used a getter for this, if u have a grocerylist class.

E.g

```
public class GroceryList {
    private ArrayList<String> groceryList = new ArrayList<String>();
```

here the ArrayList is private so all the changes or initialization have to be done using getter and setter and methods(Encapsulation) .

```java
public  String findItem( String searchItem){
     boolean exist= groceryList.contains(searchItem); //commented
    int position = groceryList.indexOf(searchItem);//-1 Not found
    if(position>=0){
        return groceryList.get(position);
    }
    return null;
}
```

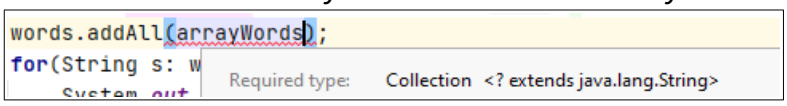
Create an interface for user, dont ask for user to write like java for e.g findItem('sahil'); So use a switch and create another method using the original method for e.g findItem. Or having all the methods in the same class, n making not-user friendly methods private and user friendly methods as public again Encapsulation.

e.g

```java
public void removeItem(int position){
    String item=groceryList.get(position);
    groceryList.remove(position);
}
```

Just remember

1.Make all the methods which are not user friendly private.
2. Create class GroceryList that is user-friendly I.e for an array for user indexing starts from 1.
3.Try to avoid print Statements in GroceryList cuz user maynot want that, return a ArrayList(with different type of items, so that user can decide how to print them). Just though of mine.
----Lec 167 time 11:00
-So the Array class also provides an alist method that is used to return a list view of the elements in the array. In fact it uses an ArrayList.




```java
Set<String> words = new HashSet<>();
String sentence = "one day in the year of the fox";
String[] arrayWords = sentence.split( regex: " ");
words.addAll(Arrays.asList(arrayWords));
for(String s: words){
    System.out.println(s);
}
```

words.addAll() takes a Collection object, so we create an List object using Arrays.asList() method white takes arrays.

How to use static method eye opener

```
private static void addNewContact(){
        System.out.println("Create a new contact.\n Enter name:");
        String name=scanner.nextLine();
        System.out.print("Enter phone no");
        int phone_no=scanner.nextInt();
        scanner.nextLine();
        Contact contact=new Contact(name,phone_no);
        mobilePhone.addNewContact(contact);
}
```

```
private static void addNewContact(){
        System.out.println("Create a new contact.\n Enter name:");
        String name=scanner.nextLine();
        System.out.print("Enter phone no");
        int phone_no=scanner.nextInt();
        scanner.nextLine();
        Contact contact=Contact.createContact(name,phone_no);
        mobilePhone.addNewContact(contact);
}
```

Above

```
// we use the reference to the contact class,
// but we havent created a new instance
// we have use it directly
public static Contact createContact(String name, int phone_no){
    return new Contact(name,phone_no);
}
```

Above is a static method, and we create a Contact object using Contact constructor.

Another **Tip:**

```
}
else{
    // in multi-user world, the record could have
    //deleted since the time we retrieve
    System.out.print("Error updating record");
}
```