**Is a Relationship**

->Dog is a Animal

->Inheritance forms is a relationship.

**Has a Relationship:**

->Object of one class used in other class, it’s called Has a relationship

->It can be one-one, one-many, many-many.

->School has students

Class Student{

Course English;

List<courses>subjects;

}

Class Courses{

List<student>whoAllTookEnglishCourse;

}

**Aggregation:**

->Ending of one object will not end of other object. Weak relation.

->If student is removed then School wont be removed.

**Composition:**

->End of one object will end of another object. Strong relation.

->If School is removed, then classes also removed.

**Java Basics Overview:**

->Java is a platform independent language.

->Java is an OOPS

->**Abstraction**: Hiding the internal implementation to the end users.

->**Encapsulation**: Wrapping of data members and data functions under a single unit.

->**Inheritance**: Child class inheriting the properties and behaviour of its parent class. Uses extends keyword.

->**Polymorphism**: Having many forms.

->**Static or overloading or compile time polymorphism**: Same name with differ in parameters.

->**Dynamic or overriding or runtime polymorphism**: Child class overriding the same method of its parent class.

->It is a highly **portable** language: Write once and run anywhere with the help of the binary. Byte code will be generated once we compile the java file.

->JVM<JRE<JDK

->Java code 🡺 Compiler 🡺 Byte Code 🡺JVM 🡺Machine code 🡺CPU 🡺Output

->**JVM** (Java virtual machine)

1. Platform Dependent

2. It converts Byte code into Machine code

3. JVM has (JIT) Just in Time Compiler which helps in conversion.

->**JRE** (Java run time environment)

1. JVM + class libraries

2. Eg: Arrays.sort(arr);

3. If we have only JRE we can run any java program, but we cannot code it.

->**JDK** (Java development kit)

1. Supports programming language.

2. Contains compiler and debugger.

3. JRE + Programming + compiler + debugger.

->JVM, JRE and JDK are platform dependent.

->**Public static void main** (String args[])

Main function is the entry point of the program, it is public so can be accessible from anywhere and static because we don’t have to create an object to call, in this case JVM calls the main method to convert byte code into machine code.

**Memory Management:**

->JVM manages the memory.

->Two types of memory 1. Stack 2. Heap

**Stack Memory:**

A screenshot of a computer program

Description automatically generated

**Heap**:

A screenshot of a computer error

Description automatically generated

IDE: Integrated Development Environment.

Java is \*\*platform independent\*\* means it can run on any machine.

LTS: Long Term Support (Java 17)

\*\*JDK\*\*[Java Development Kit] : Compile code and produces byte code.

\*\*JVM\*\*[Java Virtual Machine]: JVM is platform dependent, so it requires byte code. It is a part of JRE.

\*\*JVM\*\* is responsible for creating objects. It runs the code.

\*\*JRE\*\*[Java Runtime Environment]: It contains libraries to support java. JRE will be installed in OS.

CLI supports \*\*jshell\*\* to run java code in console.

- >\*\*JVM\*\* (Java virtual machine)

1. Platform Dependent

2. It converts Byte code into Machine code

3. JVM has (JIT) Just in Time Compiler which helps in conversion.

- >\*\*JRE\*\* (Java run time environment)

1. JVM + class libraries

2. Eg: Arrays.sort(arr);

3. If we have only JRE we can run any java program, but we cannot code it.

- >\*\*JDK\*\* (Java development kit)

1. Supports programming language.

2. Contains compiler and debugger.

3. JRE + Programming + compiler + debugger.

\*\*JDK>>JRE>>JVM\*\*

```mermaid

graph LR

Java--> Compiler

Compiler-->Bytecode

Bytecode-->JVM

JVM-->JRE

```

\*\*Variable\*\*: It is a temporary storage of data.

→In Java by default it will take double instead of float, so to define float need to add f at the end of a value say,

Ex: float f = 3.5f;

\*\*Literals\*\*: These are the constant values assign to a variables.

```mermaid

graph TD

DataTypes--> Primitive

Primitive-->Integer

Primitive-->Float

Primitive-->Character

Primitive-->Boolean

```

\*\*Type Conversion and Casting\*\*

b=(byte)a —>casting

a=b —>Conversion

If a exceeds byte range then compiler will do modulo operator with its range and stores it in variable.

\*\*Type Promotion\*\*:

```java

byte a=10;

byte b=30;

int result = a\*b; //it will automatically convert into int as it cross its range.

```

Object Oriented Programming:

Object - Properties and has behaviour.

To create an object[Instance of a class] need to have a class[Template of an object].

\*\*Method Overloading and memory\*\*

num is instance variable

a,b,c are local variables

Objects are stored in Heap memory

Local variables are stored in stack

Instance variable are part of Heap memory.

Method is stored in a Heap.

![Untitled](https://prod-files-secure.s3.us-west-2.amazonaws.com/c0551b71-959d-4a66-be07-58886cedade3/c962df83-12e1-45d2-9d70-eaa4c60cfa8e/Untitled.png)

```java

class Calculator{

int num;

public int add(int a, int b, int c){

return a+b+c;

}

public int add(int a, int b){

return a+b;

}

public double add(double a, int b){

return a+b;

}

}

```

\*\*Array\*\*:

→It is an object created in Heap memory

```java

int nums[] = new int[4]; // All the values are 0

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

int nums[] = new int[3][4];

Math.random();

int nums[][] = new int[3][] //Jagged array that no of columns is not fixed

nums[0] = new int[2];

nums[1] = new int[3];

int len = nums.length;

class Student{

int rollNo;

String Name;

double marks;

}

Student s[] = new Student[3]; // Array to hold 3 student object

s[0] = new Student();

s[1] = new Student();

```

Drawbacks of Array:

→Size is fixed, say we cannot expand the size of the array.

→Different type of datatypes cannot be added in array

\*\*foreach\*\*:

```java

for(int i:arr){

System.out.println(i);

}

for(int i[] : arr){

for(int j : i ){

System.out.println(i);

}

}

```

\*\*String\*\*: Array of char

It is a class in java

```java

String name = new String("name");

name.charAt(0);

name.concat("name2");

name.length();

String s1="Name";

String s2="Name";

// Here there is only one object for String

s1==s2 --> return true as it points to the same object referring with same data.

```

\*\*Mutable String\*\* - Something can be changed, \*\*StringBuffer\*\*

\*\*Immutable\*\* - Cannot be changed, here String are Immutable

String Buffer —> Thread safe

```java

StringBuffer sb = new StringBuffer("Name");

sb.length();

sb.append("name2");

sb.deleteCharAt(2);

sb.insert(0,"temp");

sb.substring(start,end);

String s = sb.toString();

```

StringBuilder —>Not threadSafe

\*\*Static\*\*: The value is same for all objects.

If the value in object changes, then all the objects value will be changed in case of static variable.

Static variables should be called with Class names. These variables are stored in heap memory.

```java

static String name; //common to all object

```

\*\*Inheritance\*\*: It means child[ child, subclass, derived] class getting the features of parent[Parent, superclass, base] class.

Java supports single and multilevel inheritance.

Note:

→ Java doesn’t support multiple inheritance.

→ Every class extends Object class, if a class extends to other class, in derived class constructor super() will be called.

```java

class Calc{

public int add(int a, int b){

return a+b;

}

public int sub(int a, int b){

return a-b;

}

}

class AdvCalc extends Calc{

public int mul(int a, int b){

return a\*b;

}

public int div(int a, int b){

return a/b;

}

}

class SuperAdvCalc extends AdvCalc{

public int power(int a, int b){

return (int)Math.pow(a, b);

}

}

```

\*\*Encapsulation\*\*: Wrapping up the things in a single unit or a class. Other class members can not directly access the variables or method inside the class.

```java

class Human{

public String name;

public String phoneNum;

private String aadhar;

public void setAadhar(String num){

aadhar = num;

}

public String getAadhar(){

return aadhar;

}

}

```

\*\*Method Override:\*\*

```java

class A{

public void show(){

System.out.println("In A show");

}

}

class B extends A{

public void show(){

System.out.println("In B show");

}

}

```

\*\*Boxing\*\*:

→Conversion of primitives into objects.

```java

public class first {

public static void main(String[] args) {

int num = 7;

Integer num0 = new Integer(num); //boxing

Integer num1 = num; //auto-boxing

int num2 = num1.intValue(); //unboxing

int num3= num1; // auto- unboxing

System.out.println(num2);

}

}

```

\*\*Abstract\*\*:

→Abstract method contains only the method declaration.

→Abstract method present only in abstract class.

→We cannot create an object of abstract class.

→If the class implements the abstract class, and if its not defined all the abstract methods then that particular class also a abstract class.

```java

abstract class A{

public abstract void drive();

public void playMusic()

{

System.out.println("Playing Music");

}

}

class B extends A{

public void drive()

{

System.out.println("In Drive mode");

}

}

```

\*\*Inner Class/ Anonymous inner class:\*\*

→abstract class object can be created only if we use anonymous inner class.

```java

class A{

int age;

public void show()

{

System.out.println("In A show");

}

class B

{

public void config()

{

System.out.println("In config");

}

}

}

public class first {

public static void main(String[] args)

{

A obj = new A();

A.B obj1 = obj.new B();

obj1.config();

// Anonymus inner class

A obj = new A(){

public void show(){

System.out.println("In new show");

}

};

}

}

```