**Index**

1. Java

- Introduction (Page no- 2)

- Features of JAVA (Page no- 2)

- Lifecycle of JAVA (Page no- 2)

- Some basic questions (Page no- 3)

- Wrapper Classes (Page no- 3)

- Data Types (Page no- 4)

- Reference Data Type (Page no- 5)

- My Certificates for JAVA (Page no- 5)

1. OOPs in Java

- Introduction (Page no- 6)

- OOPs concept in JAVA (Page no- 7)

- Encapsulation (Page no- 7)

- Abstraction (Page no- 8)

- Polymorphism (Page no- 8)

- Inheritance (Page no- 9)

- My Certificates for OOPs in JAVA (Page no- 12)

**1. JAVA**

*Introduction*

Java was developed by Sun Microsystems company by James Gosling in 1995 (27 years ago). First name of Java was Oak. Famous quote of java:

*“Write/Compile Once, Run Anywhere”*

JDK 🡪 Java Development Kit

JRE 🡪 Java Runtime environment

JVM 🡪 Java Virtual environment

|  |  |  |
| --- | --- | --- |
| **JDK** | **JRE** | **JVM** |
| It is a software development kit used to develop applications in java. It also contains some additional tools in addition to JRE. | It is used to execute applications written in JAVA programming. | JVM is a virtual machine. It executes JAVA bytecode and provides a runtime environment for executing Java byte code. |
| Prime functionality is development | Creating environment for code execution. | Bytecode to machine code (Interpreter) |
| Contain tools for developing, debugging and monitoring java applications. | No tools. Contains class libraries and other supporting files that JVM requires to run the program. | Does not include software development tools. |
| JDK=JRE+Development tools | JRE=JVM+libraries to run application. | JVM=only runtime environment for executing the JAVA byte code. |

Whenever we install JDK, JRE and JVM gets installed with it. There is no need to install that separately.

For example: - We have already installed a java version in a window 64 and have written a code in it and compiled the code in that window OS. Now, if we have JRE in Linux then we can execute the same Java code in Linux OS. We have JRE inside JVM.

Java IDE🡪 IDE stands for Integrated Development Environment. Eclipse and netbean.

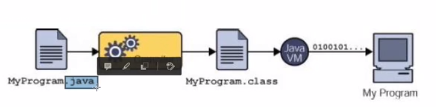
*Features of JAVA*

1. Object-oriented
2. Platform Independent
3. Simple
4. Secure
5. Architectural-neutral – Java compiler generates an architecture-neutral object file format executable on many processors, with the presence Java Runtime System.
6. Potable
7. Robust
8. Multi-threaded (do many tasks simultaneously)
9. Interpreted
10. High performance
11. Dynamic

*Lifecycle of JAVA*

1. Edit – Programmer **writes program** and stores program on disk.
2. Compile – Compiler creates **byte code** from program (.**class file**)
3. Load – **Class loader** stores **byte codes** in **memory.**
4. Execute – **Interpreter**: translates **byte codes** into **machine language**.

The **compiler** of java called as javac converts **source code** into an **intermediate file** known as Bytecode file. The **Bytecode** is unique for all types of OS means bytecode is **platform independent**. Then **Interpreter** of java converts **Bytecode** into specific **OS compatible machine code**.



*Some Basic Questions*

*Why we use both compiler and interpreter*

At the time of C and C++, only compiler was there that converts source code into specific OS machine code. The machine code was OS dependent that varied from OS to OS. Designing different compiler for different OS was time consuming. A unique compiler produces platform independent bytecodes and the specific JVM or interpreter will convert bytecodes into machine code that will vary from OS to OS.

*What is bytecode file?*

Bytecode is just like a shorthand language that store each keyword of java as a sign and each sign takes **1 byte of memory in RAM**. Hence the name called as Bytecode file.

*What is Assembler, Interpreter and Compiler?*

*Assembler* is used to convert the assembly code into the machine code. In JAVA, it is only used in case of chip level programming.

*Interpreter* is used to convert and run the code line by line.

*Compiler* is used to convert whole code into machine level code at once and then run it.

*What is a High-Level Language?*

It is a programming language with strong abstraction from the details of the computer.

|  |  |
| --- | --- |
| **High Level Language** | **Low Level Language** |
| Programmer friendly language | Machine friendly language |
| Less memory efficient | High memory efficient |
| Easy to understand | Tough to understand |
| portable | Non-portable |
| Need compiler and interpreter for execution | Need assembler for execution |

*What we need to set the environment path for java and jre while installation?*

The path is the most important environment variable of the Java environment which is **used to locate the JDK packages** that are used to convert the java source code into the machine-readable binary format. Tools like javac and java can be used by setting the path.

*Wrapper Classes*

The wrapper class implements the technique to convert the **primitive into object and object into primitive.** The eight primitive data types byte, short, int, long, float, double, char and boolean are not objects, **Wrapper classes are used for converting primitive data types into objects**, like int to Integer, long into Long etc. The process of converting primitive data types into an object is called **boxing**.

**Autoboxing-** Compiler converts primitive into wrapper object automatically

**Unboxing-** Java Unboxing is the reverse process of Autoboxing. The process to convert the wrapper class object into its corresponding primitive data type is called Java Unboxing.

**Advantages-**

1. Wrapper classes are used to provide a mechanism to ‘wrap’ or bind the values of primitive data types into an object. This helps primitive types act like objects and do the activities reserved for objects like we can add these converted types to the collections like ArrayList, HashSet, HashMap, etc.
2. As the wrapper classes have objects, we can store null as a value. We could not store null in variables of primitive datatype

***Code to understand creation of Wrapper class object***

Package css corp;

public class WrapperDemo

{

public static void main(String[] args)

{

**Integer** myInt = 10; //Autoboxing

**Double** myDouble = 11.65;

**Character** myChar = 'T';

char value=myChar; //Unboxing

**Boolean** myBool= true;

System.out.println(myInt);

System.out.println(myDouble);

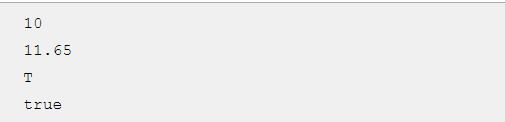
System.out.println(myChar);

System.out.println(myBool);

}

}

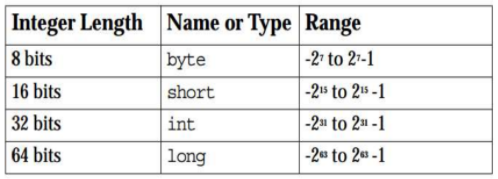
***Output***



*Data Types*

There are 8 primitive data types in Java

* Textual – char (16 bit Unicode character, ‘ ’, ‘\t’- tab character, ‘\u????’- ???? 4 hexadecimal digits)(minimum value ‘\u0000’ max value ‘\uffff’)
* Logical – Boolean (True or false, 1 or 0) – one bit of information
* Integral – byte, short, int, long (**default data type of literals is int and default value is 0**)



* Floating – float-32 bits, double-64 bits **(double is default data type for decimal values)**

*Reference Data Types*

Reference variables are created using defined constructors of the classes. **Class objects**, and various type of array variables come under **reference data type**. Default value of any reference variable is null.

A reference variable can be used to refer to any object of the declared type or any compatible type.

Eg: - Animal animal =new Animal(“giraffe”);

*My Certificates for JAVA*

Links: - <https://olympus1.greatlearning.in/course_certificate/PBXTVRTC>

- <https://olympus1.greatlearning.in/course_certificate/JMCHJKYV>

**OOPs in JAVA**

*Introduction*

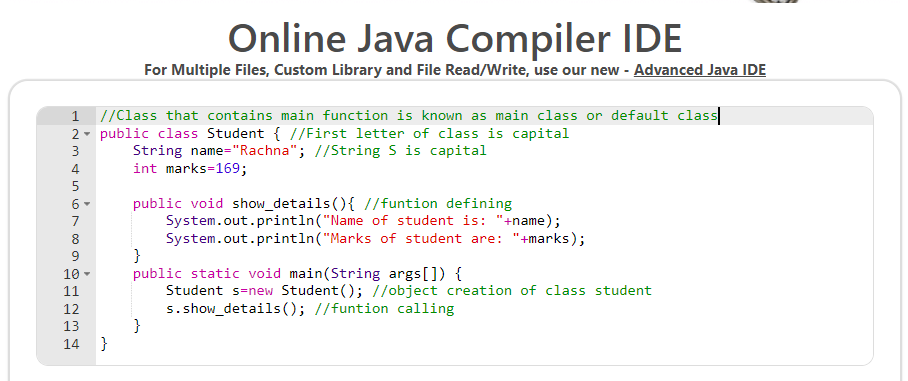
OOP stands for Object Oriented Language. It is a programming technique that relies on the concept of classes and objects. It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects.

Class is a template/blue-print for real-world entities(object). It is a user-defined datatype.

Object consists of *Properties/characteristics* and *Behavior.*

* Attributes are properties of class.
* Methods are behavior of class.

Objects are specific instances of a class. Instance means an example of something.



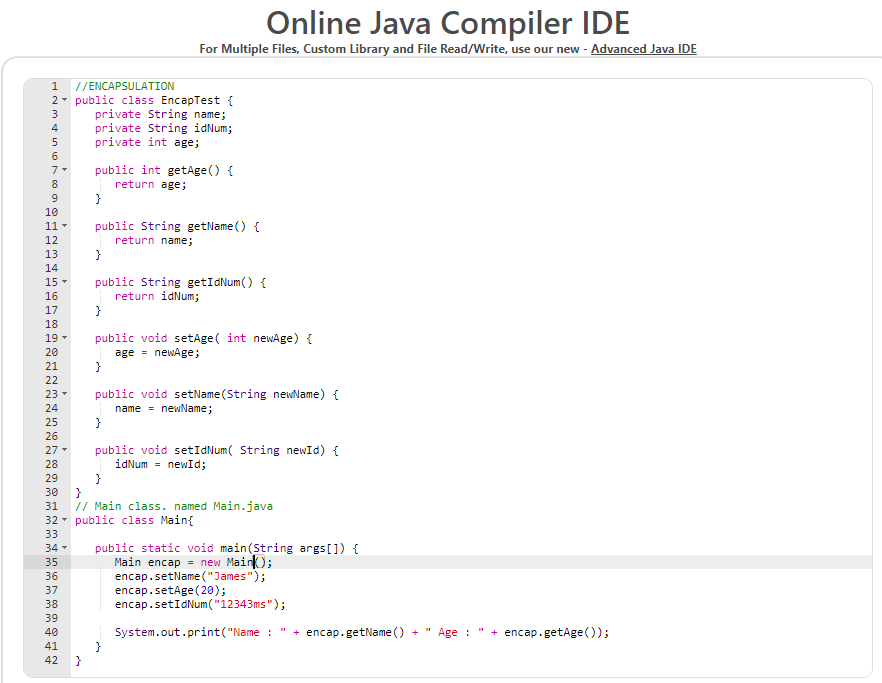


*OOPs concept in JAVA*

1. Encapsulation
2. Abstraction
3. Polymorphism
4. Inheritance

*Encapsulation*

Encapsulation in Java is a mechanism of **wrapping the data (variables) and code acting on the data (methods)** together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding. The fields of a class can be made read-only or write-only.



Output –

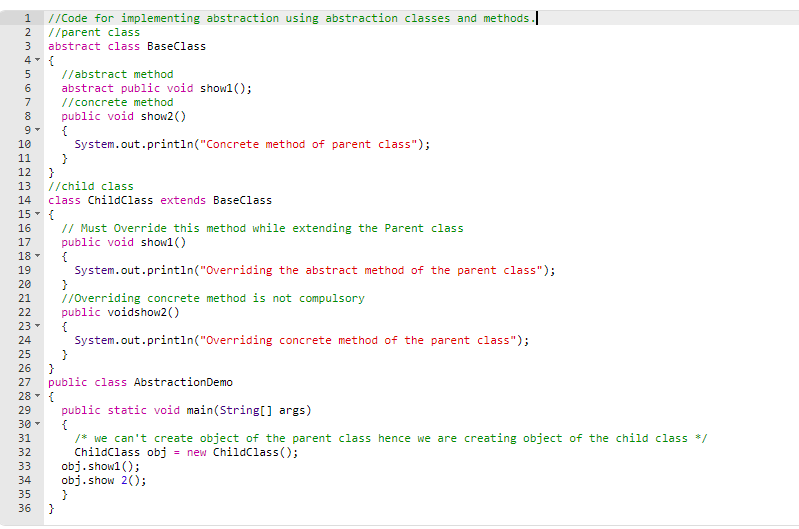


**To achieve encapsulation in Java −**

* Declare the variables of a class as **private**.
* Provide **public setter and getter methods** to modify and view the variables values.

*Abstraction*

Hiding unnecessary information from the user. In Java, we can achieve Data Abstraction using **Abstract classes** and **interfaces**.



Output: -



*Polymorphism*

It means multiple forms.

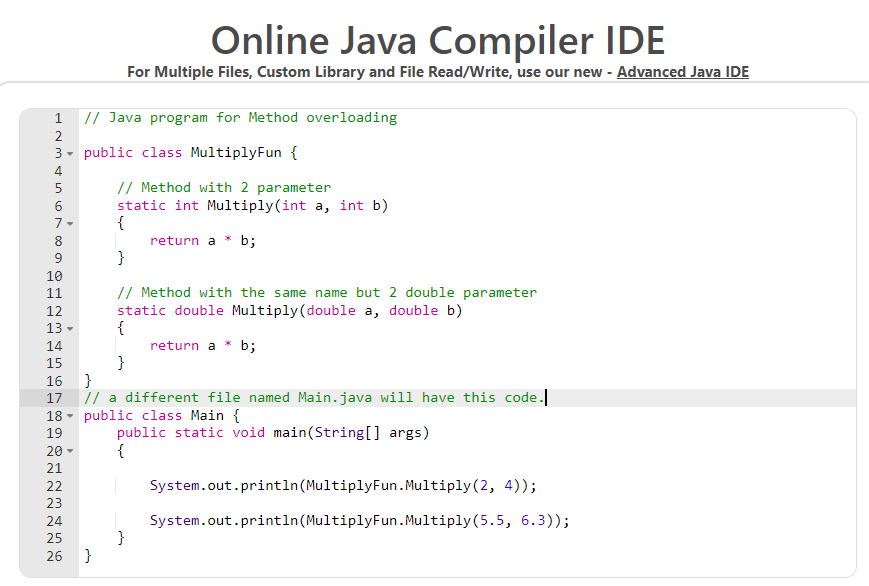
For example: - A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So, the same person possesses different behavior in different situations. This is called polymorphism.

In JAVA: - Polymorphism allows us to define one interface and have multiple implementations.

***Types of polymorphism***

1. **Compile time polymorphism** – Achieved by function overloading.

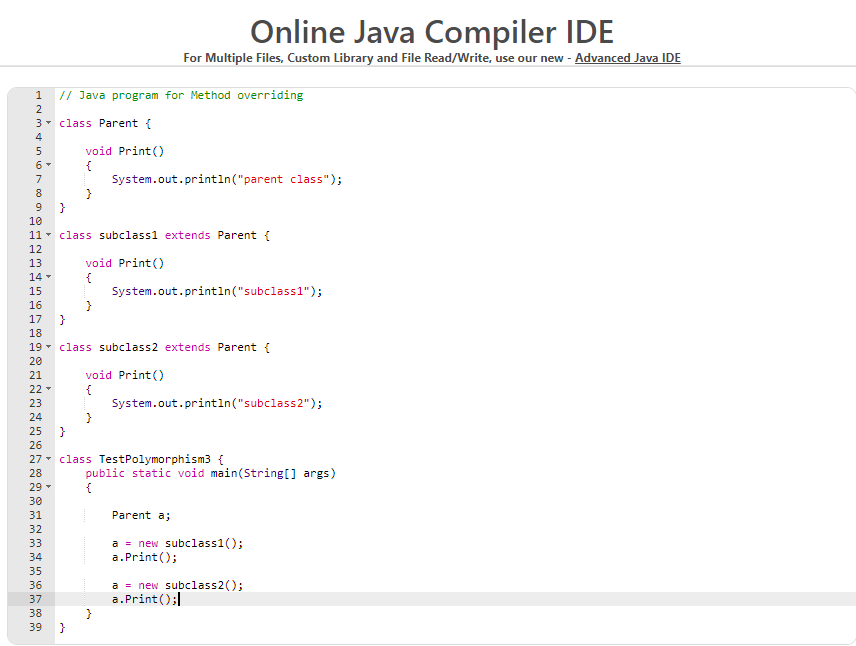
When there are multiple functions with same name but different parameters then these functions are said to be overloaded.



Output –



1. **Runtime polymorphism –** Method overriding. It occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be overridden. In this inheritance is required and name of two to three functions are same.



Output –



*Inheritance*

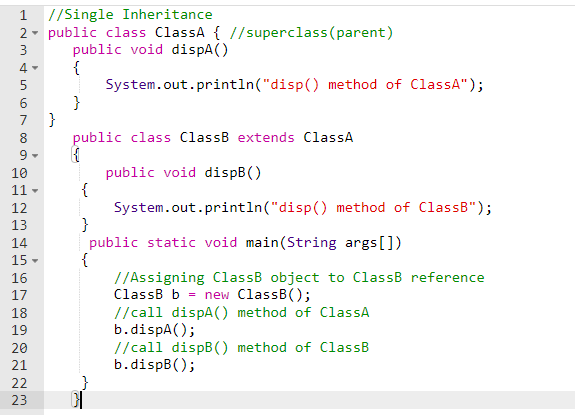
One class can derive the properties of another class.

Keyword: - extends

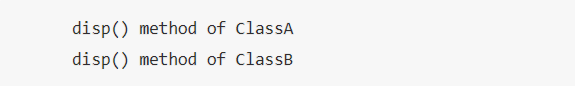
Eg:- Car extends Vehicle

Supported inheritance – Single, Multilevel, Hierarchical, Multiple(through interface) and Hybrid inheritance (through interface)

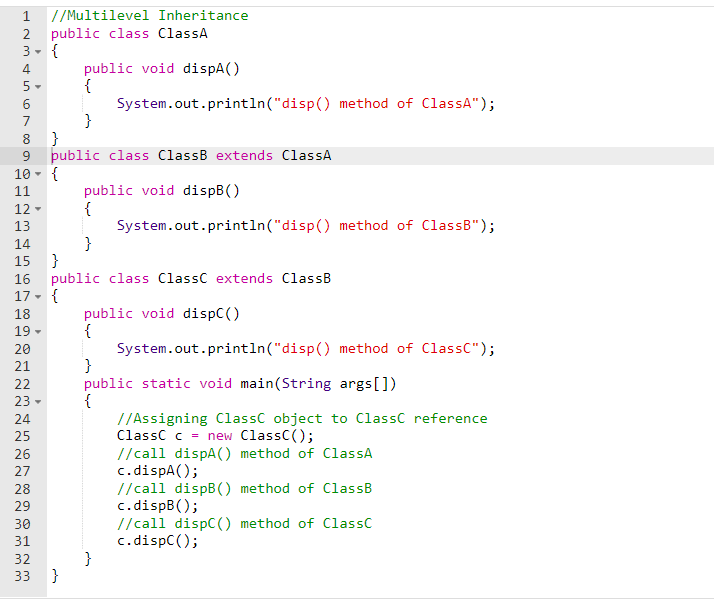
**Single Inheritance –** when class extends only single or one class.



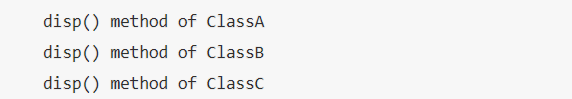
Output: -



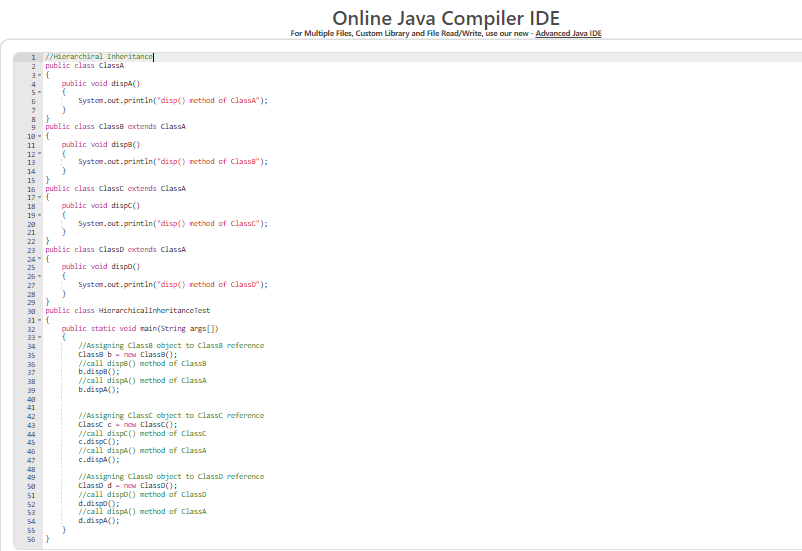
**Multilevel Inheritance –** Derived class acts as a parent to another class.



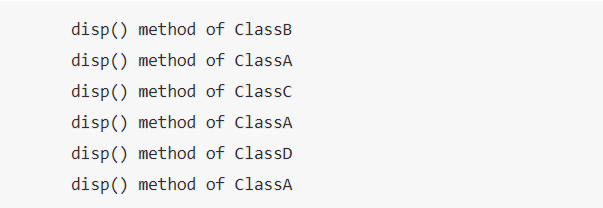
Output: -



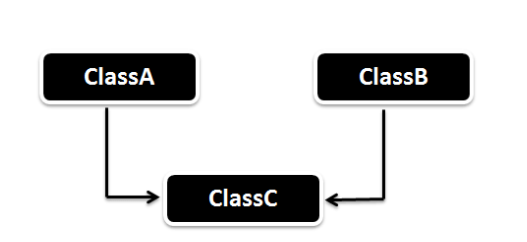
**Hierarchical Inheritance –** One parent class and multiple subclasses



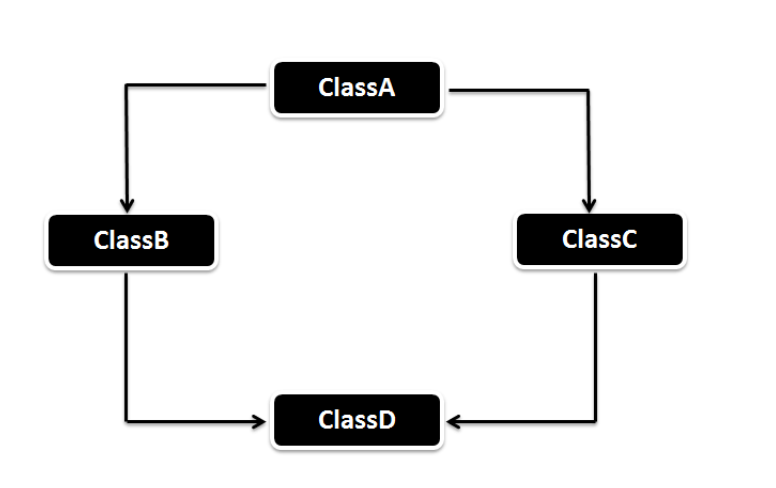
Output: -



**Multiple Inheritance -** when class extends multiple classes. Java does not support multiple inheritance as child has to manage the dependency of multiple parents on it.



**Hybrid Inheritance –** combination of single and Multiple inheritance

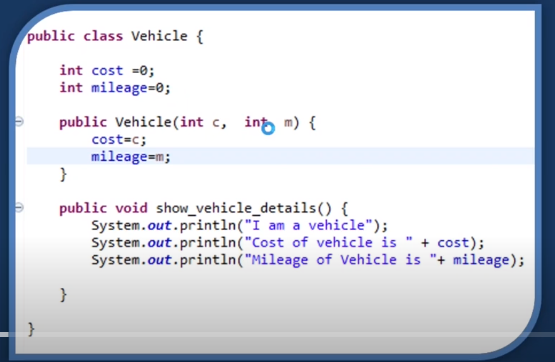
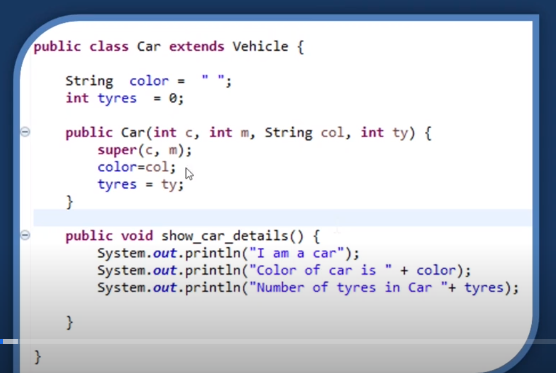


**Super keyword**

Super keyword refers to superclass(parent) objects. It is used to call the superclass methods and to access the superclass constructor. The most common use of the super keyword is to eliminate the confusion between superclass and subclasses that have methods with the same name.

For example: -

*Main class Child class*

**Final keyword**

* It is used to restrict changing value of variable.
* Restrict function overriding
* Restricts Inheritance

*My Certificates for OOPs in JAVA*

Link: - <https://olympus1.greatlearning.in/course_certificate/FNDZUAWH>

