Super Keyword

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

this keyword

this is a reference variable that refers to the current object on which the method or constructor is being invoked

Usage of Java super Keyword

1. Using this() to invoke the current class constructor
2. this is used to refer current class instance variable

final keyword

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable (Constant veriable)
2. method (you cannot override it.)
3. class (Can't be Inherite)

Java static keyword

The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block

Static keyword in Java - Javatpoint

StringBuffer :

StringBuffer is a class in Java that represents a mutable sequence of characters. It provides an alternative to the immutable String class, allowing you to modify the contents of a string without creating a new object every time.

The StringBuffer class in Java provides methods for manipulating strings. Unlike the String class, which creates immutable strings, StringBuffer is mutable, meaning its contents can be modified after it is created.

SYNTAX : StringBuffer sb = new StringBuffer("str");

Constructors of StringBuffer class :

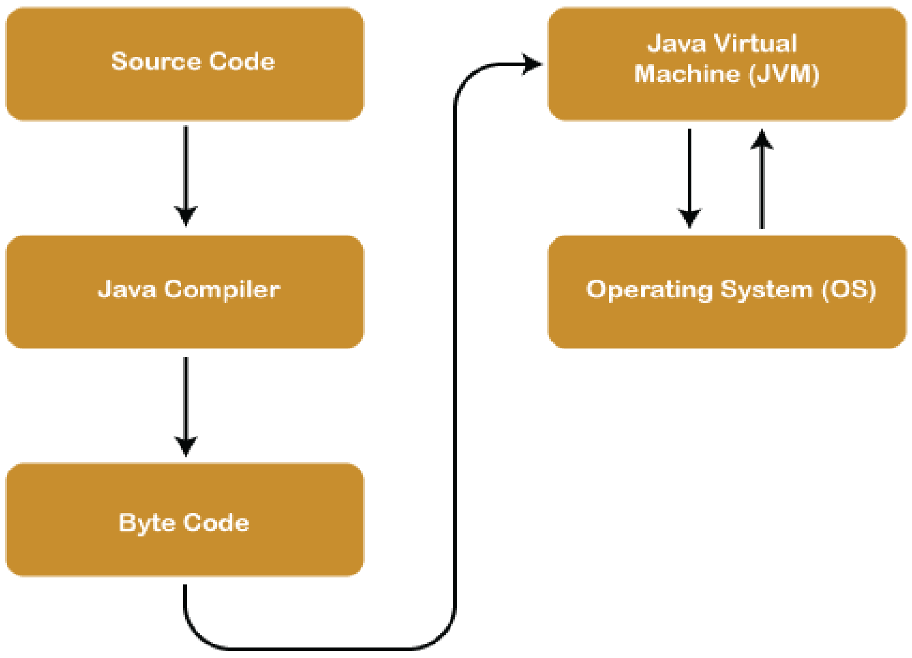
- **StringBuffer():** It reserves room for 16 characters without reallocation
- **StringBuffer(int size):** It accepts an integer argument that explicitly sets the size of the buffer.
- **StringBuffer(String str):** It accepts a string argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.

Methods :

- append()
- insert(position, str)
- delete(position, position)
- replace(position, position ,str)
- reverse()
- capacity()

Java Architecture

Java Architecture is a collection of components, i.e., JVM, JRE, and JDK. It integrates the process of interpretation and compilation. It defines all the processes involved in creating a Java program. Java Architecture explains each and every step of how a program is compiled and executed.



JDK :

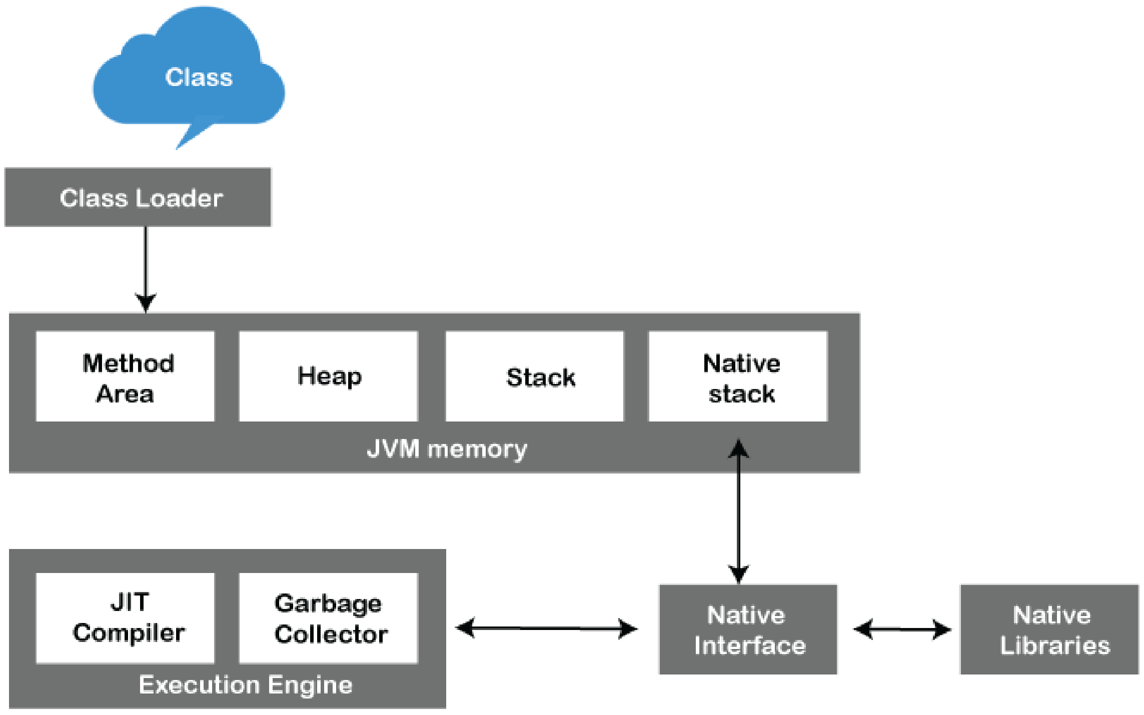
It is a software development environment used in the development of Java applications and applets. Java Development Kit holds JRE, a compiler, an interpreter or loader, and several development tools in it. To learn more about the Java Development Kit

JRE :

It provides an environment in which Java programs are executed. JRE takes our Java code, integrates it with the required libraries, and then starts the JVM to execute it.

JVM :

The main feature of Java is WORA. WORA stands for Write Once Run Anywhere. The feature states that we can write our code once and use it anywhere or on any operating system. Our Java program can run any of the platforms only because of the Java Virtual Machine. It is a Java platform component that gives us an environment to execute java programs. JVM's main task is to convert byte code into machine code.



Java Bitwise Operators :

- | bitwise OR (Return 1 if any one bit is one) 9(1001) | 10(1010) = 11(1011)
- & bitwise AND (Return 1 if both are 1) 9(1001) & 10(1010) = 8(1000)
- ^ bitwise exclusive or (if both same than 0 else 1) 9(1001) ^ 10(1010) = 3(0011)
- ~ bitwise complement (inverse all bits) ~9(1001) = (0110)
- << left sift (sift bit left side) 9(1001)<<1 = 18(10010)
- >> Right sift (remove bits from right) 9(1001)>>1 = 5(100)

Typecasting in Java

Typecasting in Java is the process of converting one data type to another data type using the casting operator.

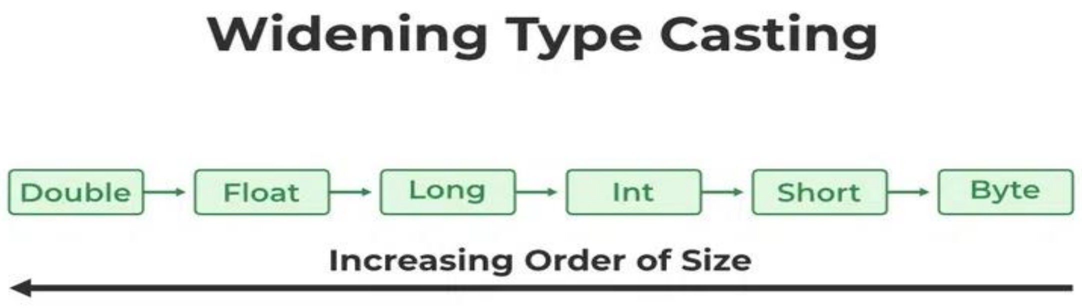
Types of Typecasting :

- Widening Type Casting
- Narrow Type Casting

Widening Type Casting :

A lower data type is transformed into a higher one by a process known as widening type casting. Implicit type casting and casting down are some names for it. It occurs naturally. Since there is no chance of data loss, it is secure.

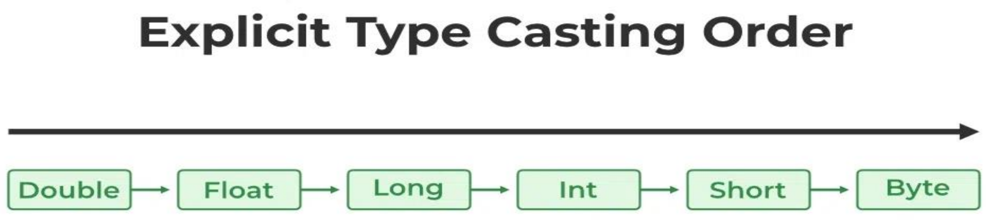
```
int i = 10;
long l = i;
double d = i;
```



Narrow Type Casting :

The process of downsizing a bigger data type into a smaller one is known as narrowing type casting. Casting up or explicit type casting are other names for it. It doesn’t just happen by itself. If we don’t explicitly do that, a compile-time error will occur. Narrowing type casting is unsafe because data loss might happen due to the lower data type’s smaller range of permitted values.

```
double i = 100.245;
short j = (short) i;
int k = (int) i;
```



Inheritance

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

Types of inheritance in java

- **Single Inheritance**
- **Multilevel Inheritance**
- **Hierarchical Inheritance**

Why multiple inheritance is not supported in java?

→ To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Polymorphism in Java

Polymorphism in Java is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms. We can perform polymorphism in java by method overloading and method overriding.

Types of polymorphism

1. Runtime Polymorphismt
2. compile-time polymorphism

Features of Java

- Simple
- Object Oriented
- MultiTasking
- Portable
- Platform Independent
- Secured
- High Performance
- Dynamic
- Support Documentation

In Java, **literals** are the constant values that appear directly in the program. It can be assigned directly to a variable. Java has various types of literals. The following figure represents a literal

Eg. Int a=500

String Methods :

- charAt(int index)
- concat(String anotherString)
- equals(String anotherString)
- split(character)
- toUpperCase() and toLowerCase()

Abstraction :

Abstraction in Java is the process in which we only show essential details/functionality to the user. The non-essential implementation details are not displayed to the user.

Simple Example to understand Abstraction:

Television remote control is an excellent example of abstraction. It simplifies the interaction with a TV by hiding the complexity behind simple buttons and symbols, making it easy without needing to understand the technical details of how the TV functions.

In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

Abstract Class :

When there is one or more abstract method in a class then that class should be declared as abstract class. It may have both abstract and non-abstract methods(methods with bodies)

- To declare a class as abstract, use abstract keyword in front of class keyword.

- We cannot create an object of abstract class, it is only used for inheritance purpose.

Abstract Methods :

Any class that contains one or more abstract methods must also be declared abstract. Abstract method can not have a body.

If a non-abstract class extends an abstract class, then the class must implement all the abstract methods of the abstract class else the concrete class has to be declared as abstract as well.

```
abstract class Subject {
        Subject() {
        System.out.println("Learning Subject");
        }

        abstract void syllabus();  //Abstract Method can not have a body

        void Learn(){
                System.out.println("Preparing Right Now!");
        }
}

class IT extends Subject {
void syllabus(){
        System.out.println("C , Java , C++");
}
}
```

```
class GFG {
        public static void main(String[] args) {
                Subject x=new IT();

                x.syllabus();
                x.Learn();
        }
}
```

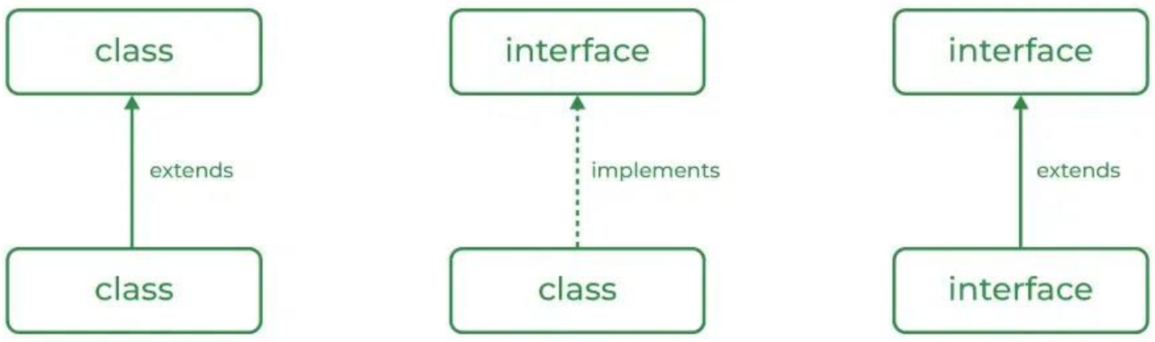
Interface :

The interface in Java is a mechanism to achieve abstraction. It is used to achieve abstraction and multiple inheritance in Java.

It cannot have a method body.

To declare an interface, use the interface keyword. It is used to provide total abstraction. That means all the methods in an interface are declared with an empty body and are public and all fields are public, static, and final by default. A class that implements an interface must implement all the methods declared in the interface. To implement the interface, use the implements keyword.

Relationship between Class and Abstract class :



Example :

```
interface In1 {

        // public, static and final
        final int a = 10;

        // public and abstract
        void display();
}

// A class that implements the interface.
class TestClass implements In1 {

        // Implementing the capabilities of
        // interface.
        public void display(){
        System.out.println("Geek");
        }

        // Driver Code
        public static void main(String[] args)
        {
                TestClass t = new TestClass();
                t.display();
                System.out.println(a);
        }
}
```

Exception:

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions. Exception handling is a mechanism in programming languages to deal with these exceptional situations gracefully, rather than letting the program crash or produce incorrect results.

Types of Exception :

1. Checked Exceptions
2. Unchecked Exceptions

Throw:
The throw keyword in programming languages is used to explicitly throw an exception. When a particular condition is met or when an error is encountered, you can use throw to create and throw an instance of an exception class.

Throws:
The throws keyword is used in method signature to declare the exceptions that the method might throw. When a method is declared with throws, it means that the method may throw one or more exceptions during its execution, and the caller of the method must handle these exceptions or propagate them further. For example:

```
        public void readFile() throws IOException {
           // code that may throw IOException
```

```
}
```

User Defined Exception :

```
class myexception extends Exception {
   myexception(String msg) {
                 super(msg);
   }
}

class error {
   int age;

   error(int age) {
      this.age = age;
   }

   void qwe() throws myexception {
      if (age < 18) {
         throw new myexception("Error aa gyi");
      } else {
         System.out.println("Error kon aayi");
      }
   }
}

class main {
   public static void main(String a[]) {
      try {
         error e = new error(1);
         e.qwe();
      } catch (myexception e) {
         System.out.println(e.getMessage());
      }
   }
}
```

Tokens:

Tokens are the smallest elements of a program that is meaningful to the compiler. They are also known as the fundamental building blocks of the program.

1. Keywords
2. Identifiers
3. Constants
4. Special Symbols – [ ] { } , ; = *
5. Operators
6. Comments - // ans /* */
7. Separators - ;