**Hybrid Inheritance:**

It is a mix of two or more types of inheritance.

Example:-

**public** **class** Parent {

**void** display() {

System.***out***.println("Parent");

}

}

**public** **interface** Mother {

**void** display();

}

**public** **class** Child **extends** Parent **implements** Parent2{

**void** display() {

System.***out***.println("Child");

}

**public** **void** disp() {

System.***out***.println("Mother");

}

}

**public** **class** GrandChild {

**void** display() {

System.***out***.println("Grand Child");

}

}

**class** Daughter **extends** Parent {

**void** display() {

System.***out***.println("Daughter");

}

**public** **static** **void** main(String[] args) {

Daughter daughter =**new** Daughter();

daughter.display();

}

}

**str.concat()**

-> concat() method is method to combine two strings.

-> In concat() method, takes only one argument of string and concatenate it with another string.

-> concat() method takes arguments of string type only.

->concat() takes concatenates two strings and return new string object only string length is greater than 0, otherwise it returns same object.

->In concat() method raises NullPointer Exception when string is concatenated with null .

->concat() method is better than + operator because it creates a new object only when the string length is greater than zero(0), so it uses less amount of memory.

**+ operator**

1. + operator used to concatenate any number of strings.
2. In + operator takes any number of arguments and combines all strings.
3. + operator takes any type of argument and converts it to string type and then combine them.
4. + operator creates a new string object every time irrespective of length of string.
5. + operator concatenates string with without any error.
6. + operator always creates a new string irrespective of length of string therefore it takes more memory.

**native**

1. The native keyword is used to declare a method as native. It means that method implementation is present in the different language.
2. A native method in Java is a method whose implementation is written in other languages such as c and c++.
3. The ‘native’ keyword is used as a method to indicate that it is implemented in another language.

**Difference between equals() and == operator**

In general, both equals() and “==” operator in Java are used to compare objects to check equality but here are some of the differences between the two: 

1. The main difference between the .equals() method and == operator is that one is a method and the other is the operator.
2. We can use == operators for reference comparison (address comparison) and .equals() method for content comparison. In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.
3. If a class does not [override the equals method](https://www.geeksforgeeks.org/overriding-equals-method-in-java/), then by default it uses the equals(Object o) method of the closest parent class that has overridden this method. See [this](https://www.geeksforgeeks.org/override-equalsobject-hashcode-method/)for detail

Example

public class Test {

    public static void main(String[] args)

    {

        String s1 = "HELLO";

        String s2 = "HELLO";

        String s3 =  new String("HELLO");

        System.out.println(s1 == s2); // true

        System.out.println(s1 == s3); // false

        System.out.println(s1.equals(s2)); // true

        System.out.println(s1.equals(s3)); // true

    }

}

[**Built-in Exceptions**](https://www.geeksforgeeks.org/built-exceptions-java-examples/)

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

1. **ArithmeticException**  
   It is thrown when an exceptional condition has occurred in an arithmetic operation.
2. **ArrayIndexOutOfBoundsException**  
   It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
3. **ClassNotFoundException**  
   This Exception is raised when we try to access a class whose definition is not found
4. **FileNotFoundException**  
   This Exception is raised when a file is not accessible or does not open.
5. **IOException**  
   It is thrown when an input-output operation failed or interrupted
6. **InterruptedException**  
   It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.
7. **NoSuchFieldException**  
   It is thrown when a class does not contain the field (or variable) specified
8. **NoSuchMethodException**  
   It is thrown when accessing a method which is not found.
9. **NullPointerException**  
   This exception is raised when referring to the members of a null object. Null represents nothing
10. **NumberFormatException**  
    This exception is raised when a method could not convert a string into a numeric format.
11. **RuntimeException**  
    This represents any exception which occurs during runtime.
12. **StringIndexOutOfBoundsException**  
    It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string