**this\_super\_final\_finally\_finalize\_exception\_handling\_throw\_throws\_keyword & static**

**this keyword:**

**Q. "this" keyword can be use in two ways.**

1. this . (this dot)

2. this() (this off)

**Q. Why do we use this keyword?**

* this can be used to refer current class instance variable.
* this can be used to invoke current class method (implicitly).
* this () can be used to invoke current class constructor.
* this can be passed as an argument in the method call.
* this can be passed as argument in the constructor call.
* this can be used to return the current class instance from the method.

**Notes:**

* The scope of "this" keyword is within the class. it always points to the current class object.
* The main purpose of using "this" keyword in real life application is to differentiate variable of class or formal parameters of methods or constructor.
* It is highly recommended to use the same variable name either in a class or method and constructor while working with similar objects.
* So, we are using this keyword to distinguish local variable and instance variable.
* If any variable is preceded by "this" JVM treated that variable as class variable.

**super keyword:**

* Super keyword in java is a reference variable that is used to refer parent class object.

**Q. Usage of super keyword:**

* super can be used to refer immediate parent class instance variable.
* super can be used to invoke immediate parent class method (overridden method).
* super () can be used to invoke immediate parent class constructor.

**Q. When need of super keyword?**

* Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features.
* and JVM gets an ambiguity. In order to differentiate between base class features and derived class features must be preceded by super keyword.
* **Final is a keyword, finally is a block and finalize is a method.**

**\*\*\*\*\*\*\*\*final keyword\*\*\*\*\*\*\*\***

**The final keyword in java is used to restrict the user.**

**Q. Usage of Final keyword.**

* Final Keyword at Variable Level---to prevent value change.
* Final Keyword at Method Level----to prevent method overriding but not inherit
* Final Keyword at Class Level-----to prevent inheritance that means can’t be extended.

**Q. What is blank or uninitialized final variable?**

Final variable which is not initialized during declaration, it’s called blank or uninitialized final variable. We must initialize the blank final variable only in constructor of the class otherwise it will throw a compilation error.

final int fees.

**Q. Can we inherit final method? Yes, but we can't override.**

**Q. Can we declare a constructor final?**

No, because constructor is never inherited.

**Note: All variables declared in an interface are by default final.**

**“finally block”**

**Q. Usage of finally block.**

* Execute code after try-catch, no matter if there is an exception or not.
* A finally block must be associated with a try block, you cannot use finally without a try block.
* You should place those statements in this block that must be executed always.
* Finally block is optional, a try-catch block is sufficient for exception handling, however if you place a finally block then it will always run after the execution of try block.
* In normal case when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.
* The statements present in the finally block execute even if the try block contains control transfer statements like return, break or continue.
* For each try block there can be zero or more catch blocks, but only one finally block.
* Finally block will not be executed if program exits (either by calling System.exit).

**"finalize method"**

* finalize is a method that is called by JVM during garbage collection.
* finalize is something related to garbage collection. Finalize is used to perform for cleanup memory processing.
* Inside the memory, there are lot of garbage is there, lot of objects is there with doesn’t have references.

public class FinalizeConcept {

public void finalize() {

System.out.println("finalize method");

}

public static void main(String[] args) {

FinalizeConcept fConcept = new FinalizeConcept();

FinalizeConcept fConcept2 = new FinalizeConcept();

fConcept = null;

fConcept2 = null;

System.gc();

}

}

**Q. What is exception handling?**

The Exception Handling in Java is one of the powerful mechanisms to handle the runtime errors so that the normal flow of the application can be maintained. Exception means unwanted behavior.

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

statement 9;

statement 10;

Suppose there are 10 statements in a Java program and an exception occurs at statement 5; the rest of the code will not be executed, i.e., statements 6 to 10 will not be executed. However, when we perform exception handling, the rest of the statements will be executed. That is why we use exception handling in Java.

**Q. Types of Exception.**

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

**1) Checked Exception**

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

**2) Unchecked Exception**

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

**3) Error**

Error is irrecoverable. Some examples of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

**Java provides five keywords that are used to handle the exception. Try, catch, finally, throw, throws**

**1. Java try block is used to enclose the code that might throw an exception. It must be used within the method.**

try{

//code that may throw an exception

}catch(Exception\_class\_Name ref){}

try{

//code that may throw an exception

}finally{}

**Java catch block is used to handle the exception by declaring the type of exception within the parameter.**

**Throw:**

* throw an exception deliveredly or I want to create my own exception and I want to throw it.
* Sometimes what happens, some exceptions are not occurred in Java.
* Java can't create each and every exception. That time I can create my own exception by throw keyword.
* it created by "add throws declaration" or "surround with try-catch block".

**Throws:**

* used to handle the exception but we don't report anything, there is no option for report anything.
* The "throws" is used to throw an exception for a method

**Q. What is the need of having throws keyword when you can handle exception using try-catch?**

We already know we can handle exceptions using try-catch block. The throws does the same thing that try-catch does but there are some cases where you would prefer throws over try-catch. For example: Let’s say we have a method myMethod() that has statements that can throw either ArithmeticException or NullPointerException, in this case you can use try-catch. But suppose you have several such methods that can cause exceptions, in that case it would be tedious to write these try-catch for each method. The code will become unnecessary long and will be less readable. One way to overcome this problem is by using throws like this: declare the exceptions in the method signature using throws and handle the exceptions where you are calling this method by using try-catch.

**Q. Why use nested try block?**

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

**Q. What is call by value and call by reference?**

There is call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The change being done in the called method, is not affected in the calling method.

**STATIC\_KEYWORD**

* Static keyword is used for memory management.
* It makes the program more efficient by savings memory.
* Static keyword can be used with
  1. Variables (also known as class variable)
  2. Methods (also known as class method)
  3. Blocks and
  4. Nested Classes.
* Static members belong to the class instead of specific instance, this means if you make a member static, you can access it without creating object.
* Static variable is related to the Object, its related to the class. So, it’s also called Class variable.
* A static method means that it can be accessed without creating an object of the class, unlike public.
* Static block is used for initializing the static variables.

**Q. When and why, we use static variable?**

Suppose we want to store record of all employees of any company, in this case, employee id is unique for every employee but company name is common for all. When we create a static variable as a company name then only once memory is allocated otherwise it allocates a memory space each time for every employee.

**Q. Restrictions of Static key word.**

There are two main restrictions for the static method. They are:

* The static method cannot use non static data member or call non-static method directly.
* This and super cannot be used in static context.

**Q. Why a static block executes before main method**?

A class has to be loaded in main memory before we start using it. Static block is executed during class loading. This is why a static block executes before the main method.

**Q. Difference between static and final key word.**

* Static keyword always fixed the memory that means that it will be located only once in the program whereas the final keyword always fixed the value that means it makes variable values constant.

**Note: As for as real-time statement there concern every final variable should be declared as static but there is no compulsion that every static variable declared as final.**

**Q. Can we overload a static method?**

Yes. But method signature must be different.

**Q. Can we override a static method?**

**No,** we cannot override static methods because method overriding is based on dynamic binding at runtime and the static methods are bonded using static binding at compile time. So, we cannot override static methods.

The calling of method depends upon the type of object that calls the static method. It means:

* If we call a static method by using the parent class object, the original static method will be called from the parent class.
* If we call a static method by using the child class object, the static method of the child class will be called.

In the following example, the ParentClass has a static method named display () and the ChildClass also has the same method signature. The method in the derived class (ChildClass) hides the method in the base class. let's see an example.

https://www.javatpoint.com/can-we-override-static-method-in-java