Static:

=======

\* Static is a keyword.

\* Static is a modifier.

\* A member prefixed with static is called as static member.

\*Static members are also known as class members.

Static members:

=============

\* Static variable

\* Static initializer

\* Static method

Non-static member:

================

A member, which is not prefixed with static, is called as non-static member.

Characterstics:

============

\* Non-static members belongs to an object. Therefore, it is also called as instance members.

\* Memory is allocated inside the object.

\* We can access them with the help of object reference

\* We can create any number of copies of a non-static variable.

\* Memory will be allocated once per every object created.

Non-static members:

=================

Non-static variable

Non-static method

Non-static initializer

Constructor

|  |  |
| --- | --- |
| Static variable | Non-static variable |
|  |  |
| Static variable are prefixed with static keyword | Non-static variable are not prefixed with static keyword |
| Memory is allocated in CSA | Are allocated memory inside the Object |
| We can access them with class name | We can access them with object reference |
| Only one copy of static variable is created | We can create multiple copies of non-static variable |
| Initialized with default value during loading  process of class | Initialized with default value during loading process of  object |

|  |  |
| --- | --- |
| Static method | Non-static method |
|  |  |
| Prefixed with static keyword | Not prefixed with static keyword |
| we can access them with class name | We can access them with object reference |
| The block which belongs to static is called as static context | The block which belongs to non-static is called as  non-static context |
| allocated memory in CSA | Allocated memory inside object |
| Allocated memory during the loading process of  class | Allocated memory during loading process of object |

|  |  |
| --- | --- |
| Static initializer | Non-static initializer |
|  |  |
| Executed during loading process of class | Executed during loading process of object |
| It will get executed only once | They will get executed once per every object created |
| Block should be prefixed with static keyword | We need not to prefix the block with static keyword |

|  |  |
| --- | --- |
| Multi line Initializer | Method |
|  |  |
| Multi line initializer cannot have access modifier | Method can have access modifier |
| It does not have return type | Method has return type |
| It does not have parameter | Methods can have parameter |
| Multiline initializer cannot be overridden | Method can be overridden |
| We cannot call a multi-line initializer | We can call a method |
| Multi line initializer will get executed without  calling | Method will get executed only after calling |

|  |  |
| --- | --- |
| Method | Constructor |
|  |  |
| Method name can be anything | Name should be same as class name |
| Method has return type | Constructor does not have return type |
| Method is executed to do a specific task | Constructor is executed to initialize non static members |
| Method can be prefixed with abstract keyword | Constructor cannot be abstract in nature |
| We can override methods | We cannot override constructors |
| We cannot use this() and super() statement in a  method | We can use this() and super() statement in a constructor |
| Since methods have return type we can use  return statement in a method | We cannot use return statement inside a constructor |

|  |  |
| --- | --- |
| Abstract class | Interface |
|  |  |
| We cannot achieve 100% abstraction in a class | We can achieve 100% abstraction in an interface |
| We cannot achieve multiple inheritance in  abstract classes | We can achieve multiple inheritance in interfaces |
| We can have both abstract and concrete non static methods in an abstract class | We can only have non static abstract methods in an  interface |
| Abstract classes can have non static variable | Interface cannot have non static variable |
| we can use any access modifier with non-static  methods in an abstract class | But non static methods are public by default |
|  |  |
| We should use abstract keyword with a non-static method to make it abstract | Non static methods are by default abstract in nature |

|  |  |
| --- | --- |
| this | super |
| this is a keyword which is used to call  members of same class | super is a keyword which is used to call members of parent class |
| this holds the reference of current object | super holds the reference of immediate parent |
| When we have a local variable and non-static variable with same name this is used to access the non-static variable | When we have a non-static variable with same name in parent class and child class, super is used to access the member of the parent class |

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  | |
| this() | | super() | |
| this() is used to call constructor of same class | | super() is used to call constructor of parent class | |
| this() should be added explicitly | | super() is added implicitly by the compiler | |
| When there are n constructors there can be only (n-1) constructor with this() | | When there are n constructors there can be n constructor with super() | |
| this() should be used in the same class | | super() should be used in the child class | |
|  | |  | |
| StringBuilder | | StringBuffer | |
| It is not synchronized | | It is synchronized | |
| It is not thread safe | | It is thread safe | |
| It is faster than string buffer | | It is slower than string builder | |
| It is introduced in Java 1.5 | | It is introduced in Java 1.0 | |
|  | |  | |
| throw | | throws | |
| It is used to create an exception | | It is used to declare an exception | |
| We can create one exception at a time | | We can declare multiple exception at a time | |
| It should be used in the method definition | | It should be used in the method declaration | |
| We can throw checked and unchecked exception | | We can declare checked exception only | |
|  | |  | |
| final | finally | | finalize |
| final is a keyword | finally is a block | | finalize is a method |
| final can be used with class, method and variable | finally should be used with try/try & catch block | | finalize can be called with object reference |
| final class cannot be inherited final variable cannot be reinitialized final method cannot be overridden | finally block will get executed even when there is an exception finally block will not get executed when the exception occurred outside try block | | finalize method is used to clear the garbage before the object is destroyed |
| final variable/method can be called | finally block cannot be called | | finalize method can be called |

|  |  |
| --- | --- |
| ArrayList | List |
| It is c class | It is an interface |
| We can create object for it | We cannot create object for it |
| It has concrete method | It has abstract methods |
| It has constructors | It does not have constructor |
| Parent class is object class and list interface | Parent interface is collection |
|  |  |
| List | Set |
| List is having index | Set does not have index |
| List allows duplicate | Set does not allow duplicates |
| List is ordered collection of elements | Set is unordered collection of elements |
| The implementing classes are ArrayList, Linked list etc. | The implementing classes are Hash Set and TreeSet |
|  |  |
| Class | Interface |
| We can create an object for a class | We cannot create an object for a class |
| We can have concrete method | We cannot have concrete method |
| We can have concrete non static method | We cannot have concrete non-static method |
| We can have non-static variable | We cannot have non-static variable |
| Constructors are allowed | Constructors are not allowed |
| Multiple inheritance is not possible | Multiple inheritance is not possible |
|  |  |
| Class | Object |
| Class is a user defined non primitive data type | Object is a block of memory in heap area |
| Class is a blueprint of an object | Object is an instance of a class |
| Class is a logical entity | Object is physical entity |
| Class is created with new keyword | Object is created with new keyword |
| Class is created only once | Object can be created multiple times |