Access Modifier:

Four Types in java:

Public

Protected

Default

Private

Public: if we declare class member as public, then we can access those class members:

* Within the class
* Within the package
* Also, outside the package

Protected: if we declare class member as Protected, then we can access those class members:

* Within the class
* Within the package
* Also, outside the package only in IS- A relationship [inheritance]

Default: if we declare class member as Default, the class member can accessible :

* Within the class
* Within the package

Private: if we declare class member as private, then the class member can access only inside the class.

In method overriding, the visibility of the class method should widen, should not be narrowed

Example 1: If Method in a parent class Declare as DEFAULT, then Overridden Method in the child class can be declared as DEFAULT, PUBLIC, PROTECTED .. should not be PRIVATE because visibility is narrowed for PRIVATE method which can access only inside the class.

Example 2: If Method in parent class declared as PROTECTED, then the overridden method in the child class can be declared as PUBLIC or PROTECTED, should not be DEFAULT or PRIVATE because visibility of the DEFAULT or PRIVATE is narrowed.

Example 3: If method in the parent class declared as PUBLIC then overridden method in the child class should declared as PUBLIC, should not be PROTECED, DEFAULT, PRIVATE because visibility of PROTECED, DEFAULT, PRIVATE are narrowed.

Note: PRIVATE methods cannot be overridden.

LOCAL VARIABLE:

variables which are declared inside the method is known as LOCAL VARIABLES. Which cannot access outside the method.

GLOBAL VARIABLE:

Variables which are declared outside the method is know as GLOBAL VARIABLES which can access inside the method by creating object of the class inside the METHOD.

ARRAY:

Array is a collection of multiple values of the same Data Type.

Type:1 : object Type

Syntax: <Datatype> [] variable= new <Datatype>[size]

Example: int [] a=new int [4];

Type-2: Literal creation of Array

Syntax: <Datatype> [] a= {};

Example: int [] a= {1,2.3.4.5};

Disadvantages:

1. Size of an array would be static
2. Cannot store multiple data type values in one array.

Exception: When size is exceeded, system will throw IndexOutOfBoundException

STRING:

String is class in Java.

String classes are Immutable such that the content of the string object cannot be modified.

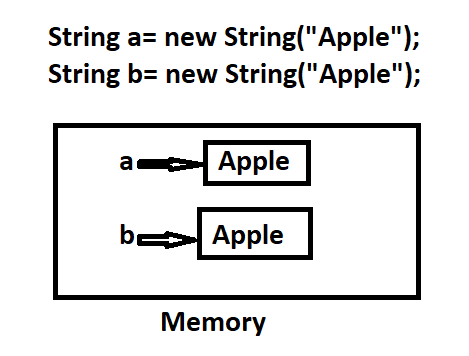
String can declare in two types;

Type1: object Type:

Syntax: <Datatype> variable= new<Datatype>(“String value”);

Example: string a= new String(“Apple”);

If we create an object by using new keyword, then compiler will create a new object in the memory even if that string is already present in the java Memory

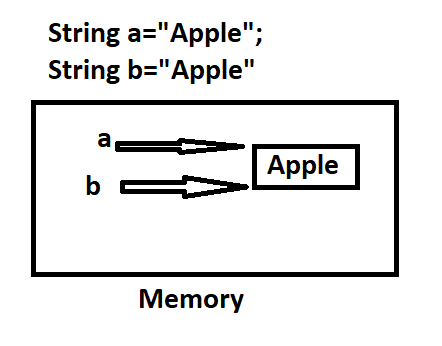


Type 2: Literal creation of string

String a= “Apple”;

String b=” Apple”;

If we declare a String using String literal, and string value is already existing in the memory, them compiler will assign newly declared String variable to the existed String value;



String Builder and String Buffer

String Builder:

String Builder is a class.

String Builder can be declared in two ways:

1) String Builder a= new String Builder (“Apple”);

2) String Builder a= new String Builder();

a. append(“Apple”);

String Builder Objects are Mutable so that String builder content can be changed or modified.

String Builder a= new String Builder (“Apple”);

a.append(“Banana”);

output: AppleBanana

String Builder cannot be declared as :

Example : String Builder a=”Apple”;

String Buffer :

String Buffer is a class.

String Buffer can be declared in two ways:

1) String Buffer a= new String Buffer (“Apple”);

2) String v a= new String Buffer ();

a. append(“Apple”);

String Buffer Objects are Mutable so that String Buffer content can be changed or modified.

String Buffer a= new String Buffer (“Apple”);

a.append(“Banana”);

output: AppleBanana

Difference between STRING vs STRING BUILDER vs STRING BUFFER

|  |  |  |
| --- | --- | --- |
| STRING | STRING BUILDER | STRING BUFFER |
| 1) String class objects are Immutable.  2) String class methods are not Thread Safe(not synchronized ) | 1) String Builder objects are Mutable.  2) String Builder methods are not -Thread safe (non- synchronized) | 1) String Buffer objects are Mutable.  2) String Buffer methods are Thread safe (synchronized) |

OOPS

Inheritance:

What is inheritance and where are you using in your automation?

a) What it is?

Inheritance is Object-oriented concept where a sub class inherit from the parent class (methods, fields). Can be applied to others like interface, annotation, enum as well.

Keyword: extends

Purpose: Reusability of existing code & to avoid code duplication

Types: Single, hierarchal, Multiple and Multi-Level

b) Java example: ArrayList extends AbstractList (Runtime Exception extends Exception)

c) Selenium example: RemoteWebDriver -> Chromium Driver -> Chrome driver /

Interface Inheritance: Search Context -> WebDriver

d) Framework example: BaseClass(ProjectSpecificMethods) -> Pages & Tests

Polymorphism: Polymorphism is Object oriented concept where single action can be transformed into multiple forms.

Two Types of polymorphism:

1) compile Time polymorphism

2) Run Time polymorphism

|  |  |
| --- | --- |
| compile Time polymorphism: | Run Time polymorphism |
| - The methods resolution is performed by the compiler based on parameter list.  - Once the compiler, perform the method resolution based on parameter list, then compiler performs static binding operation.  - In static binding the compiler will bind the method declaration to the method definition during the compilation.  - Once the static binding is completed, the reference calling overloaded method behaves in a polymorphic way.  - Hence the reference variable is considered as polymorphic reference at compile time. | -The method resolution is performed by the JVM based on the object present inside the reference variable.  -When the jvm performs methods resolution based on the object present in the reference, then JVM performs dynamic binding operation.  -The dynamic binding performed by the JVM will bind the method declaration to the method definition during runtime.  -Once the binding is completed the JVM will make the reference variable calling the method to become polymorphic.  -Hence the reference variable is considered as runtime polymorphism. |

Method overloading:

Method Overloading is also called as compile Time polymorphism or static binding.

a) where methods of a class having same name with different parameters.

b) Purpose: simplify verbose

c) Java examples: subString(int) // only starts index and subString (int, int) // start and end

System.out.println() is also an example for methodoverloading, whose job is tp print the output in the console. Println() method which takes strings, int, float etc.

d) Selenium examples: switchTo().frame

e) Framework examples: locateElement(String) -> default by xpath, locateElement(String, String) -> by any locator

Method Overriding:

Method overriding is also called as Run Time polymorphism or dynamic Binding

a) Override the super class method behavior at sub class level

b) Purpose: when like override the functionality

c) Java examples: equals method in String class overrides the Object class

d) Selenium examples: quit method in Chromium Driver overrides the quit method in RWD

e) Framework examples: BeforeMethod in registration testcase override the BeforeMethod at hooks level (default: login)

Constructor

Constructor in java is similar to method that is invoked when an object of the class is created. Unlike java methods, Constructors are the one which has same name as a class name without return type.

//Constructor are called when the object of the class is created.

Three types:

* Default constructor
* Zero Parameterized constructor (constructor without any parameter)
* Parameterized constructor (constructor with one or more parameters)

Constructor overloading : class with different number of parameters or different type of parameters.

Note: constructor cant be inherited and also cant be overridden

Why we use constructor?

Without a constructor, we cant create an instances of the class.

|  |  |
| --- | --- |
| Constructor | Methods |
| Constructor name should same as class name | Method name can have same or different name |
| Constructor does not have any return type | Method should have return type |
| Constructor cant be inherited hence cant be overridden | Method can inherited and it can be overridden |
| Constructors are called whenever object is created | Methods are called whenever the call is made to it using return variable or by the class name |
| If we don’t write any constructor then compiler writes default constructor | No default methods |
| Constructors cant be declared as Static or final or abstract | methods can be declared as Static or final or abstract |

Abstraction

The process of hiding business logics from the normal users and allowing them to use only the functionality.

Interface

1. Interface is a type of definition block which is completely abstract.
2. Using the interface, we can achieve multiple inheritance in java because it resolves the diamond problem and constructor chaining problem.
3. Interface can contain public, static, final variables and public, abstract, instance methods.
4. Interface cannot contain constructor, blocks, static concrete methods, instance concrete methods and instance variables.
5. An object of interface cannot be created.
6. An interface can inherit another interface by using keyword “extends”
7. A class can inherit an interface by using a keyword “implements”
8. An interface cannot inherit a class.

A class can implements one or more interfaces. When a class is implementing a multiple interface’s we need to override all the abstract methods of those interface or else we can declare that class also as Abstract.

**Interface :1**

**public** **interface** InterfaceA {

**public** **void** m();

}

Interface:2

**public** **interface** InterfaceB {

**public** **void** n();

}

Class:

**public** **class** Test **implements** InterfaceA, InterfaceB {

@Override

**public** **void** n() {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** m() {

// **TODO** Auto-generated method stub

}

}

A class can implement one or more interfaces and also can extends one class.

Whenever the extending and also implementing then first we need to extend the class and then implements the interface.

**Interface :1**

**public** **interface** InterfaceA {

**public** **void** m();

}

Interface:2

**public** **interface** InterfaceB {

**public** **void** n();

}

Parent class:

**public** **class** Demo {

**public** **void** write() {

System.***out***.println("parent class");

}

}

Child class:

**public** **class** Test **extends** Demo **implements** InterfaceA, InterfaceB {

@Override

**public** **void** n() {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** m() {

// **TODO** Auto-generated method stub

}

**public** **void** write() {

System.***out***.println("child class");

}

}

Note: We can achieve 100% abstraction using abstract classes but we cant achieve multiple inheritance. Whenever we want to achieve 100% abstraction we go with interfaces.

* The main class can be declared as abstract [anyways the JVM calls the main method using the class name but not by creating the object of that class]
* Abstract class can have a constructors [But we cant create the object]. These constructors can be called from the child class using “Super()”

Abstract class

1. An abstract class is a type of class which is declared with a keyword called ‘abstract’.
2. An abstract class contains both concrete methods and abstract methods.
3. A concrete method is a type of method which have method definition or method body and the method doesn’t contain as abstract keyword.
4. An abstract method is a type of method which don’t have method definition, or a method body and the method does contain abstract keyword.
5. If a class contains abstract methods then class should declare as abstract class.
6. If a class doesn’t contain abstract methods then can also class can declare as abstract class.
7. An object of the abstract class cant be created.
8. An abstract class doesn’t support multiple inheritance because of diamond problem or constructor chaining.

Why abstract class object cannot be created ?

An Abstract class can have both concrete methods and abstract methods. If we create object for abstract for Abstract class then we can generate output of concrete methods, cannot generate output for abstract methods.

Difference between Abstract Class and Interface

|  |  |
| --- | --- |
| Abstract class | Interface |
| 1.Abstract class contains both abstract methods and concrete methods. | 1. Interface contains only abstract methods. |
| 2.we cannot achieve multiple inheritance | 2.we can able to achieve multiple inheritance |
| 3.Any access modifiers can be used inside the abstract class [But abstract method should not be private] | 3. All the methods declared inside the interface are by default public abstract. |
| 4. we can declare any type of variable inside the abstract class | 4. All the variables declared inside the interface are by default public, static, final |
| 5.abstract class is used when there is some common property need to be shared among child class | 5. interface is used when there no common property need to be shared among child class |
| 6. Abstract class can contain constructor | 6.interface cant have a constructor |
| 7.A class can always extends abstract class | 7. A class should always implements as interface |

Static:

* Static is a keyword in java that can used block level/method level and variable level.
* While creating the static variable, the value of the static variable remains same
* The static variables and methods can access without creating an object.

Note: we cant declare local variable as Static

Static methods cant be overridden

this – keyword:

this is a keyword in java, this – is always point to current class. this is equal to creating an object of the current class. Whenever there is naming conflict, we always make use of ‘this’ keyword to point the global variable. Because whenever there is a naming conflict, compiler will always give preference to local variables.

‘this ’ cannot be access inside the static method

Example:

**public** **class** Baby {

String color;

**int** age;

**public** Baby(String color, **int** age) {

**this**.color = color;

**this**.age = age;

System.***out***.println(color);

}

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

Baby b = **new** Baby("pink", 20);

}

}

Super – keyword:

Super is a keyword in java, which is applicable only in inheritance, where we used to get the Super class functionality.

‘Super ’ cannot be access inside the static method

**public** **class** A {

**int** i=30;

}

**public** **class** B **extends** A {

**int** i = 60;

**public** **void** m() {

**int** i = 90;

System.***out***.println(i);

System.***out***.println("global variable value " + **this**.i);

System.***out***.println("Super class variabe value " + **super**.i);

}

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

B b = **new** B();

b.m();

}

}

final – keyword:

final is a keyword which is used to declare constants.

Final keyword can be used on – variables, class and methods.

On variables:

* Local & Static variables can be declared as final
* Static variables declared as final , called as class constants.
* Instance variable declared as final, called as object constants.

On Methods:

* if we declare any method as final, then that method can’t be overridden.
* Final methods can be overloaded
* Final methods can be inherited but can’t be overridden

On Class:

* We can declare class as final
* A final class cant be inherited
* We can create object of the final class and we can access the final class members.

Note: Abstract class & Abstract methods should not be declared as final

Encapsulation

Grouping the related set of data and binding them into one class is called Encapsulation.

Example:

* A class encapsulate states and behavior
* Group of statements are encapsulated inside a method

Encapsulation can be achieved using ‘java beans’

Rules to create bean class:

* Bean class should be a non-abstract class
* All the variables should be declared as PRIVATE
* Each private variable should have one corresponding public setter and public getter method.
* Class should have one public default constructor
* Bean class should implement a serializable interface

Note: Serializable interface is a mark up interface ,present in java.io.pacakge. Markup interface is an empty interface

**public** **class** BaseClass {

**private** String name;

**private** **int** age;

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **int** getAge() {

**return** age;

}

}

**public** **class** ChildClass **extends** BaseClass{

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

ChildClass c= **new** ChildClass();

c.setAge(20);

c.setName("max");

System.***out***.println(c.getAge());

System.***out***.println(c.getName());

}

}

**Advantages of Encapsulation**

**Data Hiding:** During this process, the user doesn’t have any idea about the inner implementation of the class, like how the values are stored inside a variable. The user only aware that they are passing a value to a setter method, and the value is initialized in that variable.

**Reusability:** The encapsulated code is more flexible and easier to change with the new requirements.

* It Prevents other classes to access the private fields.
* It also allows us to modify the implemented code without breaking other’s code. Those have already implemented the code.
* It improved the maintainability of the program and application.
* If you require to define the fields as read-only, then you can do that by similarly defining only getter methods if you want the fields as only write-only, then you can also do that by only mentioning the setter method.

Collections

* Collection framework is a readymade utility which contains ‘n’ no of class and interfaces.
* In collection framework we can deal with only non-primitives.
* A collection framework cannot store primitive datatypes. To store primitive datatypes, must convert primitive datatype to wrapper class object and store the wrapper class object in the framework
* Collection framework have two main interfaces.
* 1) collection
* 2) Map

Collection

List Set

* Arraylist linked hashset
* Linkedlist treeset
* Vector hashset

Array list:

* 1. Array list class  is a resizable [array](https://www.w3schools.com/java/java_arrays.asp), present in java.util pacakage
  2. Array list class is providing implementation for collection interface and list interface abstract methods
  3. Array list support duplication of objects and supports insertion order.
  4. Array list supports one or more null values.
  5. Array list must be used in the application to fetch the data or to search data.

Syntax:

ArrayList<Integer> al = new ArrayList<integer>();

To add values:

al.add(“johm”);

al.add(index, element)

To update:

Al.set (index, element)

Remove:

List.remove();

Linked List:

* 1. Linked list is a class present in java.utilpacakage
  2. Linked list class is providing implementation for collection interface and list interface abstract methods and queue abstract methods.
  3. Linked list supports insertion order.
  4. Linked list support duplicate order.
  5. Linked list supports null values and duplicate null values.
  6. Linked list must be used in a real time application to perform frequent insertion, updation and deletion of data.

Syntax:

LinkedList<Integer> al = new LinkedList<integer>();

|  |  |
| --- | --- |
| ArrayList | LinkedList |
| ArrayList is slow in addition & deletion of the object (because no.of swaps are more) | LinkedList is fast in addition & deletion of the object (because no.of swaps are less) |
| ArrayList is fast in iteration the values | Linked List is slow in Iterating the values |
| ArrayList don’t have any additional methods | Linked list have some additional methods like 1) add first 2) add last   1. Remove first 4) remove last 2. Get first and get last |

SET:

HashSet:

* Hashset is a class which provides implementation for collection interface abstract methods.
* Hashset stores only unique objects and does not support any kind of duplication.
* Hashset does not support an insertion order
* Supports null values but doesn’t support duplication of null values.

Set<String> s= new Hashset<String>();

Linked Hashset:

* The only difference between hashset and linked hashset is hashset doesn’t support insertion order and linked hashset support insertion order.
* Set<String> s= new LinkedHashset<String>();

TreeSet:

* Treeset is a class present in java util pacakage and treeset provides implementation for sorted list navigable set and collection
* Treeset does not support insertion order where as it arranges the objects in ascending order.
* Treeset does not support duplicate objects.

Set<String> s= new Treeset<String>();

Note:- there is no get method in set, we cant get a particular value in set.

* We cant insert a value in set
* We cant update
* We can remove

To retrieve values:

* Add the values in a set to the list

List<String> al= new ArrayList<String>(names);

* Go for for loop

For(string eachName:names)

Difference between HashMap and Hashtable

|  |  |
| --- | --- |
| HashMap | Hashtable |
| 1) HashMap is non synchronized. It is not-thread safe and can't be shared between many threads without proper synchronization code. | 1- Hashtable is synchronized. It is thread-safe and can be shared with many threads. |
| 2) We can make the HashMap as synchronized by calling this code  Map m = Collections.synchronizedMap(hashMap); | 2-Hashtable is internally synchronized and can't be unsynchronized. |
| 3) HashMap allows one null key and multiple null values. | 3-Hashtable doesn't allow any null key or value. |
| 4) HashMap is fast. | 4-Hashtable is slow. |
| 5) HashMap is traversed by Iterator. | 5-Hashtable is traversed by Enumerator and Iterator. |

Why Thread synchronization is required?

When multiple threads access the common resources, it results in data inconsistency or data corruption. To overcome this problem, thread synchronized is used.

How to achieve ?

By using the keyword called “synchronized”.

When multiple threads access a method declared with synchronized keyword the jvm will internally called a thread queue and stores as the multiple threads inside the queue. Once the threads are stored in the queue the jvm will randomly pick thread from the thread queue and allow thread to access the synchronized method. Once the thread access the synchronized method , Jvm will implicitly call the wait() of the object class and wait() will lock the resources and it doesn’t allow other threads to access the synchronized method.

Once the execution of current thread is completed the jvm uses notify() or notifyall() of the object class to indicate the threads waiting in the queue that current method is completed the execution of synchronized().

Then Jvm will pick randomly another thread and allow thread to access the synchronized() and report the above operation until thread queue becomes empty.

What is garbage collector in java

garbage means unreferenced objects.

way to destroy the unused objects.

* It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.
* It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

Throw vs throws?

|  |  |
| --- | --- |
| Throw | Throws |
| - Throw is a keyword which is used to throw only the exception object.  - Throw can throw only one exception.  - throw keyword can be used only within the method  Example:  Class A{  Void div(){  Syso(“”);  Try{  Syso(10/0);  }  Catch(Arithmetic Exception e){  Syso(“”caught exception at div);  Throw e;  }  }  } | - Throws is a keyword which is used with the method.  Example:  Class A{  Void div() throws IOExeception, SQLException  {  //code  }  }  - Throws keyword can be declared with multiple exceptions.  - |

Exception Handling:

- Exception is an event which triggered during the execution of program and disturbs the normal flow of the program.

- if the exceptions are not handled then program will be terminated automatically.

- exceptions are handled using try catch block.

- the code which might raise the exception should written inside the try block. And the catch block will catch only the exception which are raised from its try block.

- A single try can have multiple catch block but parent catch should be the last block.

- we cant have only try, only catch or only finally block.

- we can write a try block without a catch block but it has to be with finally block.

- when we have try, catch & finally , finally should be the last block.

- finally should not written in between try & catch block or in between two catch blocks.

To make this code to execute even after throwing the exception , we need to write that code inside the finally block

Final vs finalize vs finally

|  |  |  |  |
| --- | --- | --- | --- |
| Key | Final | Finally, | Finalize |
| Definition | final is the keyword and access modifier which is used to apply restrictions on a class, method or variable. | finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not. | finalize is the method in Java which is used to perform clean up processing just before object is garbage collected. |
| Applicable to | Final keyword is used with the classes, methods and variables. | Finally block is always related to the try and catch block in exception handling. | finalize() method is used with the objects. |
| Functionality | (1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited. | (1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block | finalize method performs the cleaning activities with respect to the object before its destruction. |
| Execution | Final method is executed only when we call it. | Finally block is executed as soon as the try-catch block is executed.  It's execution is not dependant on the exception. | finalize method is executed just before the object is destroyed. |

What is ENUM:

An enum is a special "class" that represents a group of **constants** (unchangeable variables, like final variables).

#### **Difference between Enums and Classes**

An enum can, just like a class, have attributes and methods. The only difference is that enum constants are public, static and final (unchangeable - cannot be overridden).

An enum cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces).

#### **Why And When To Use Enums?**

Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.