**Polymorphism**

* + Poly-many
  + Morphs-forms
* **Def:**

Single entity that acts differently for each different class

* Method-Overriding is a form of polymorphism, and it is an example of run time polymorphism.//­Iate Binding
* Method-Overloading is an example of compile time polymorphism//Early binding
* **ABSTRACTION:**
  + Hiding the implementation details by just providing the functionalities
* Can be achieved in two ways
  + Abstract Class
  + Interface

**Abstract Class**

Any class that has been declared Using the abstract keyword is known as abstract class

* Abstract Method:

Any method that has been declared with abstract keyword that method is known as abstract method

* + - Abstract method don’t have any body
* Abstract classes can have both abstract methods as well as concrete method
* Any class having an abstract method should be declared as abstract but the vice versa is not true
* Its implementation will be in child class
* We cannot create objects, but we can create constructors
* We can create concrete methods in abstract class

If a class has been declared as abstract then that class should be extended by any child class

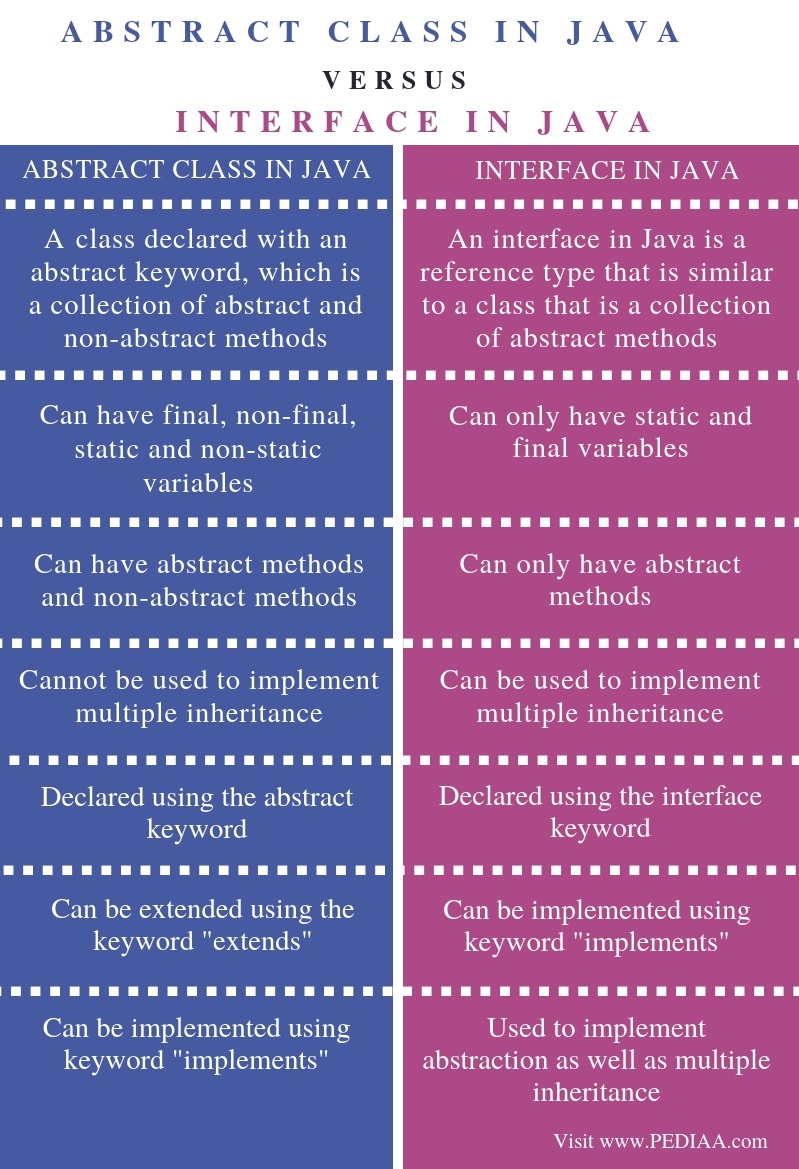
If a concrete class extending an abstract class it has to implement all the abstract methods in the abstract class otherwise that child class also has to be declared as an abstract class

**INTERFACE**

­­we can implement the interfaces using the “implements” keyword

* Without giving the abstract keyword also, all the methods are abstract and Abstract methods do not specify a body
* Constructors are not allowed
* 🡺java8 feature ,We can create concrete method with static or default keyword,but in abstract class we can create evn without those keywords
* Without static or default the keyword, that method will be taken implicitly as an abstract class

|  |  |
| --- | --- |
| **Interface** | **Abstract class** |
| Cant create concrete methods without static or default | Can create concrete methods without any keywords |
| We can implement two interface at the same time | We can`t extends two classes at the same time |
| \*By default methods will be abstract and public |  |
| ---------------------- | ----------------------------------------- |
|  |  |



|  |  |
| --- | --- |
| Interface | Abstract class |
| An interface in java is a reference type that is similar to a class that is a collection of abstract methods | A class declared with an abstract keyword which is a collection of abstract and non-abstract methods |
| can have only have static and final variables | can have only have static ,non-static, final and non-final variables |
| Can ave abstract methods |  |
|  |  |
|  |  |
|  |  |

* Interfaces are of three types:
  + - Marker
    - Typical
    - Functional

Functional:- It should have only one abstract method

To ensure this we will use 🡺“@FunctionalInterface”.

Marker:- doesn`t have any method, generally used to provide special interfaces to a class

Typical:- Having any no. of abstract or concrete methods

**Some of the pre-defined maker interfaces**

Cloneable- in clone methods🡺in java. lang

Serializable- file handling🡺in io. serializable

Remote🡺rmi. remote

Static import helps you to get all the Static methods

* All variables declared inside interface are implicitly public static final variables(constants).

**ENCAPSULATION**

Hiding the data members and provide some accessibility to those members using public methods.

DEF: Encapsulation is a mechanism with which we wrap up the data members and the function members into a single object

We can make some data variables like read-only, write only

Getter helps us to access those private data members from outside class- read only

Setters manipulates the data- write only

No static for getters and setters

*PACKAGES:-*

Separating the classes in different packages

Helps to access, delete and create the classes easily

Provides reusability

* way to access the class in another package using import statement

Dog d= **new** Dog();

Dog d1= **new** Dog();

* way to access the class in another package using Qualified name

com.dev.Constructor.Son son=**new** com.dev.Constructor.Son();

FINAL

The value of the variable which is initialized with final key word will stay like tat forever upto its life, it cannot be reinitialized

We have to give the final variable name in “BLOCKEDLETTERS”

\*final methods cant be overridden

\*final classes cannot be inherited

\* final classes can inherit the other normal classes

**OBJECTCLASS**

In java, each and every classdirectly or indirectly inherits the properties of object class i.e., object classis the super most class in java

Each and evrey class either a predefined clss or a user defined class, is a child class of object

Methods of Object class:

1. getClass(): used to get the address of that class
2. clone(): clones an object ,makes a new object but that class should implements an interface CLONABLE
3. equals(): return type is Boolean and used to compare to diff objects we have to override hashcode (ret type is int)and equals (boolean)methods together
4. toString():

//threads

1. wait():
2. wait(long):
3. wait(long ,int ):
4. notify()
5. notifyAll()

//invoked by garbage collector

1. finalize():invokes as soon as the execution of the program ends

1. hashCode();

***Strings***

* String is a reference type. As a result String variable holds a reference of an objects created in string class.
* Even though the string s are primitive types, string is the only type where operator Overloading I supported in java
* String is immutable, the immutable object is an object that once created it can`t be changed
* String pool contains two parts:
  + - 1.contsant pool :In constant pool no duplicates are allowed
    - 2.non-constant pool: where as in non-constant pool duplicates are allowed

***StringBuilder and StringBuffer Classes:***

String is a powerfull class but not very efficient

Because strings are immutable, any methods of the string classthat modifies the string in any way,must create a new StringObject.

To overcome this problem, java offers two alternatives to string class:

StringBuilder

StringBuffer,class🡪is thread safe mutable sequence of characters, assign upto 16-char

String builder and string buffer classes are mirror images .Both have the same methods and performs the same string manipulations

* The only diff b/w S.buffer is thread safe where as String buffer is not thread safe
* You cant assign a string literal directly
* Both are final classes we cant extend those

***Constructuors:***

We hav1 16 consructors in Strings

Void StringBuffer(){

Super(16)

}

-no characters init

-stores 16 chars

Void StringBuffer(int capacity){

Super(capacity)

}

-no characters init and with the specified initial capacity

-stores as per capacity

-negative array size exception 🡺if the capacity is <0

Void StringBuffer(String str){

Super(str.length+16);

Append(str);

}

-constructs a string buffer initialized to the contents of the specified string, the initial capacity of the string buffer is length of the string argument + 16

Void StringBuffer(CharSequence seq){

This(seq.length+16);

Append(seq);

}

***StringBulider***

StringBuilder-just mutable sequence of charcters

Constructors are as same as in the StringBuffer

String builder s little more efficient than String buffer

***EXCEPTION***

Is an error event that can happen during the execution of the program and disrupts the normal flow of execution.

**Exception Handling:**

**Exc**eption in java can arise from different kinds of situations such as wrong data entered by user, hardware failure, network connection Failure, database server down, etc.

Whenever an error occurs while executing the statement, creates an exception objects and then the normal flow of the execution halts

The exception object contains a lot of debugging information such as method hierarchy, line number where the execution occurred ,type of exception

Occurs at runtime, Exceptions cant fix compile time errors

* The process of creating the exception object and handling it over to the run time is known as “throwing Exception”
* If an appropriate exception handle is found ,exception is passed to the handler to process it. The handler is said to be “catching the exception”

DIFFERENTIATE EXCEPTIONS AND ERRORS IN JAVA

|  |  |
| --- | --- |
| EXCEPTIONS | ERRORS |
| Can be recovered | Cant be recovered |
| Two types:  Checked  Unchecked | No such classification, all errors are unchecked |
|  |  |

**THROWS:-**

**THROW:-**

**TRY-CATCH:-**

**FINALLY;-**

**Java exception hierarchy:**

Java exceptions are hierarchical and inheritance is used to categorize the diff. types of exceptions.

Throwable is a parent class of java exceptions hierarchy it has two child objects \*error

\*exception

Error:

Errors are exceptional scenarios that are out of the scope if the application and its not possible to anticipate and recover them.

For eg: hardware failure, Jvm crash, or out of memory error

Exceptions:

Checked Exceptions:

Go for try and catch block,

Complier alerts there will be the exception during the run time and fix that now, gives warning

Unchecked Exceptions:

Go for try and finally,

Wont give us warning.

* Exception follows hybrid inheritance.
  + - Constructors
      * Exception()
      * Exception(String)
      * Exception(String,Throwable)
      * Exception(Throwable)
      * Exception(String,Throwable,Boolean,boolean)

🡺Throwable class implements Serializable interface

* + - Constructors
      * Throwable ()
      * Throwable (String)
      * Throwable (String,Throwable)
      * Throwable (Throwable)

Exception(String,Throwable,Boolean,boolean)

* Public String getMessage()

**{**

**}**

* Public String

We can extend Exception to a user defined Exception

STREAMS -database connectivity

Close()-to free up space