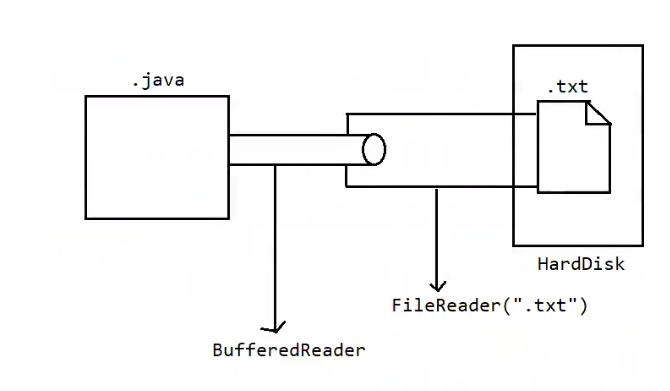
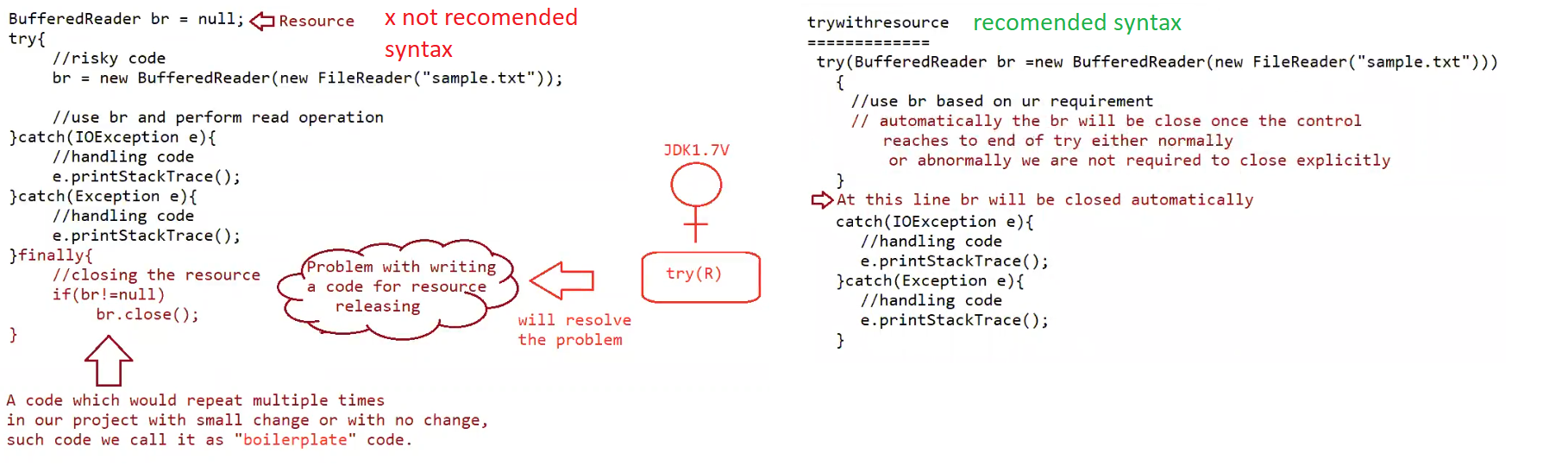
Resource : an object which you have used in your code and perform operation is called resource

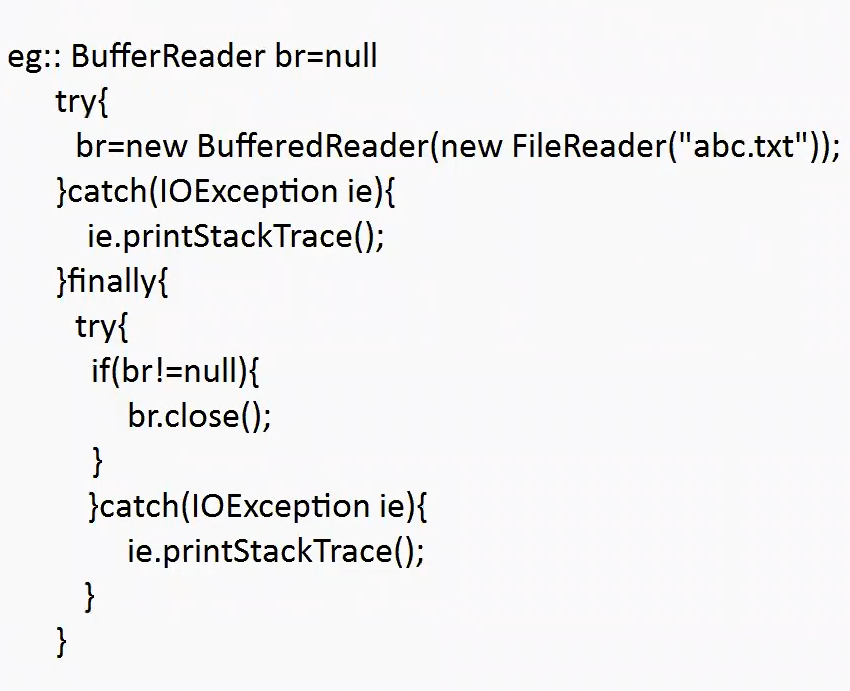
(or)

An built class for which you have created an object , using that object you called some methods to perform some operation is called resource





Until jdk 1.6 , it is compulsorily required to write finally block to close all the resources which are open as a part of try catch block

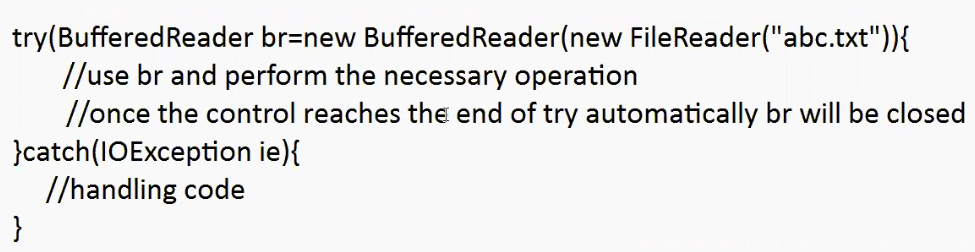


Problems in this approach :

1. Compulsorily the programmer needs to close all opened resources ,otherwise it increases the complexity of the program
2. compulsorily we should write finally block explicitly , and it should contain the code that contains closing of all the resources . this increases the length of the code and reduces readability.
3. To overcome this problem sun microsystems team introduced try – with resources in 1.7 version of jdk.

Try with resource :

* In this approach the resources which are opened as a part of try-catch block will be closed automatically once control reaches out of try block normally or abnormally , so it is not required to close explicitly and the complexity of the program will be reduced.
* It is not required to write the finally block explicitly , so length of code will be reduced and readability is improved.



Rules of writing try with resource:

1. We can declare no of resources, but all these resources should be separated with “;” semicolon.

Eg : try ( r1 ; r2 ; r3 …… ) {

// use the resources

}

1. All the resources are said to be AutoClosable resources if the class implements an interface called “ java.lang.AutoClosable” either directly (or) indirectly.

Eg: java.io package classes , java.sql.package classes

C:\Users\Administrator>javap java.lang.AutoCloseable

Compiled from "AutoCloseable.java"

public interface java.lang.AutoCloseable {

public abstract void close() throws java.lang.Exception;

}

Note : which ever class has implemented this interface those class objects are referred as objects.

1. All the resource reference by default are treated as implicitly final and hence we cant perform reassignment with in try block.

try(BufferedReader br = new BufferedReader ( new FileWriter(“abc.txt”))

{

br = new BufferedReader( newFileWriter(“abc.txt”));

}

o/p : C.E cant reassign a value

1. Until jdk 1.6 version try should compulsorily be followed by either catch (o) finally but from jdk1.7 version we can take only try with resources without catch ( or) finally.

Eg: try(R){

// valid

}

1. Advantage of try with resource concept is finally block will become dummy because we are not required to close resources explicitly.
2. try with resource nesting is also possible

syntax :

Eg: try(R1){

try(R2){

try(R3){

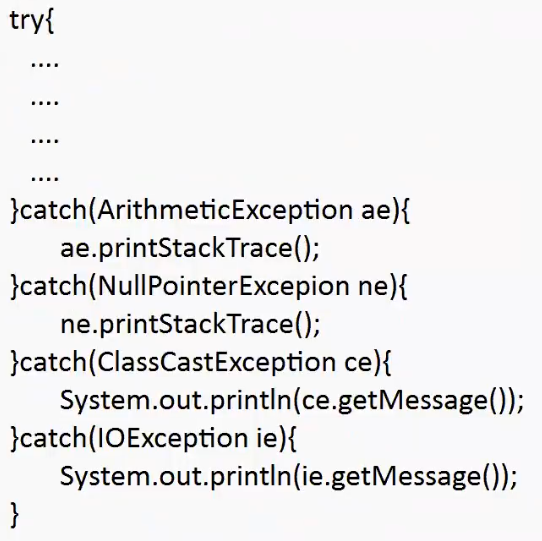
}

}

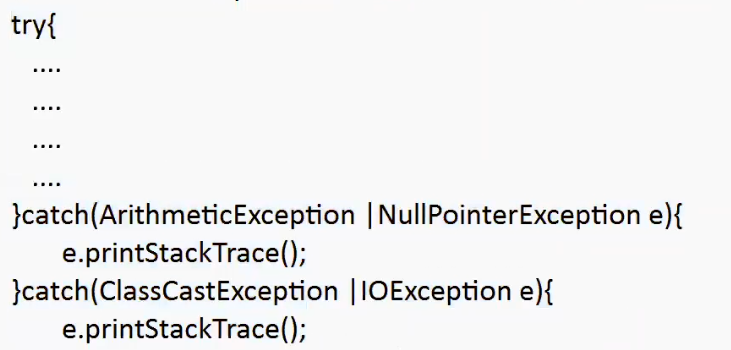
}

multi catch block :

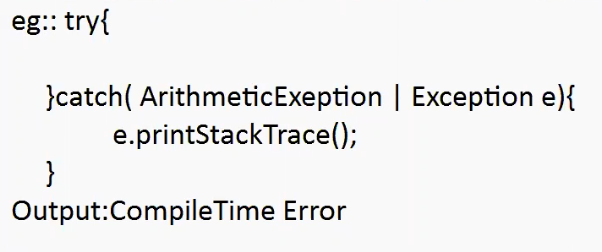
1. till jdk1.6 even though we have multiple exception having same handling code , we have to write separate catch block for every exceptions, it increases the length of the code and reviews readability

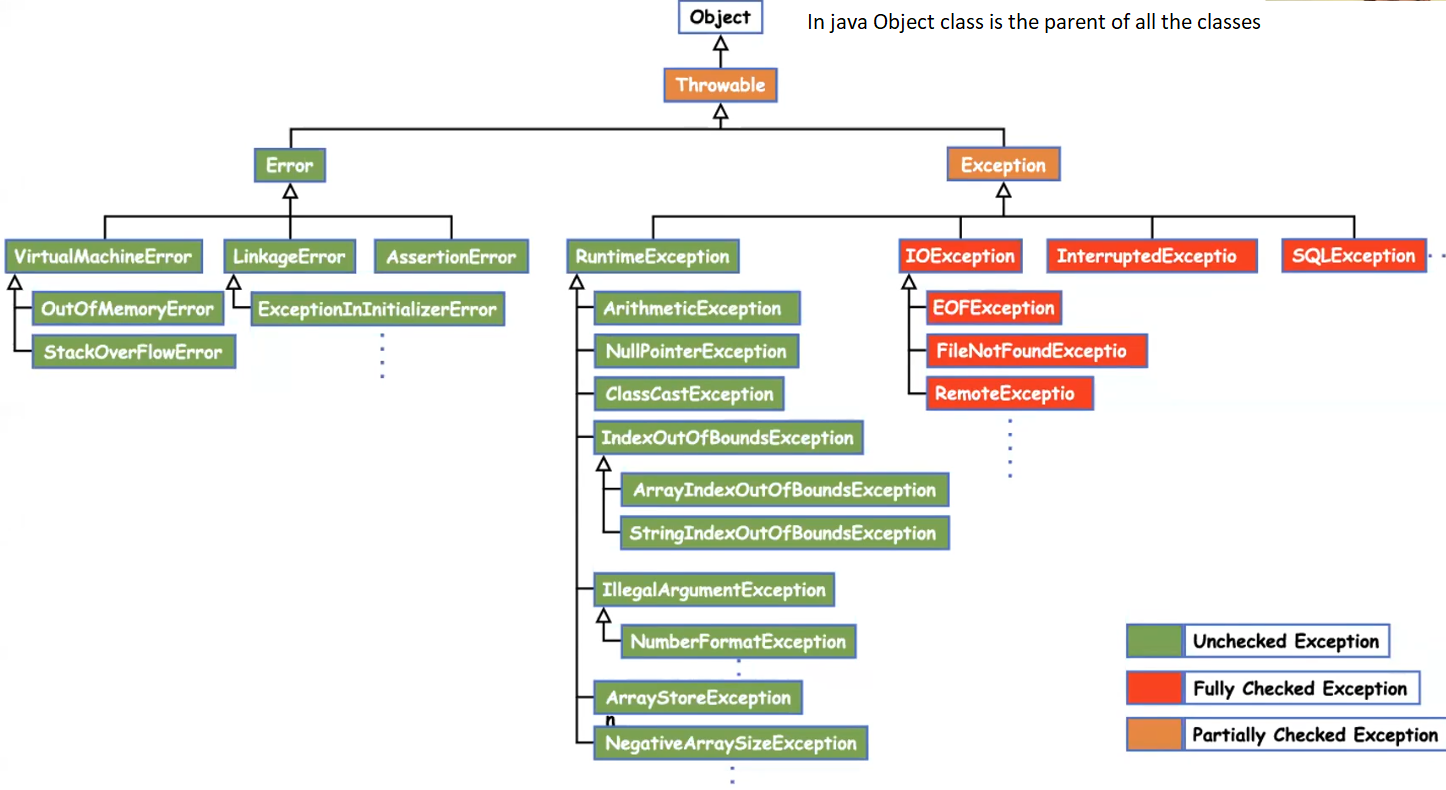


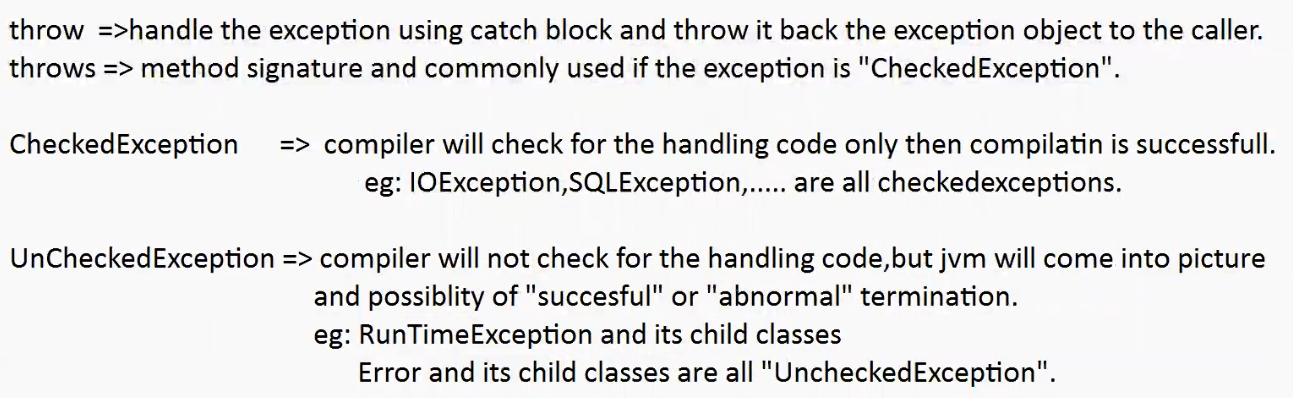
To overcome this problem the sun-microsystem team has introduced “multi-catch block” concept in version jdk 1.7



1. in multi-catch block , there should not be any relation between exception types ( either child to parent (or) parent to child ( or) same type )





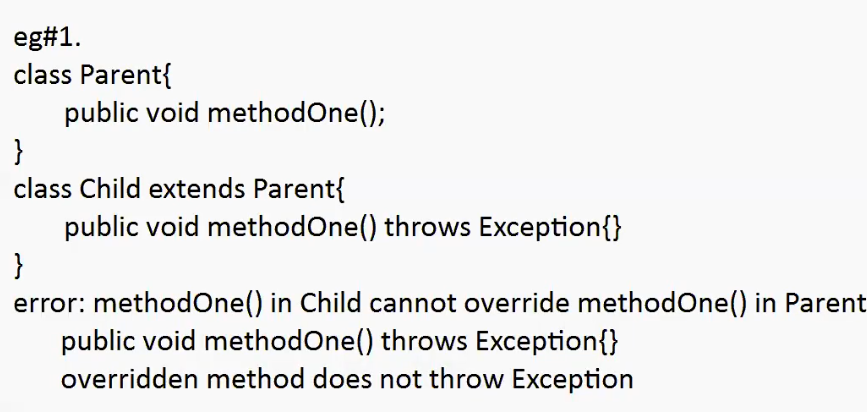


Rules of overriding when exception is involved :

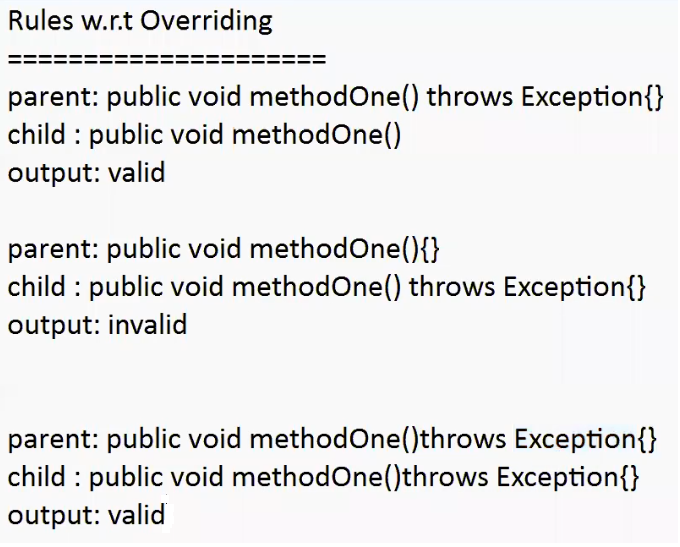
While overriding if the child class method throws any checked exception compulsorily its parent class method should throw the same checked exception Or its parent (exception ) otherwise it will lead to compile time error.

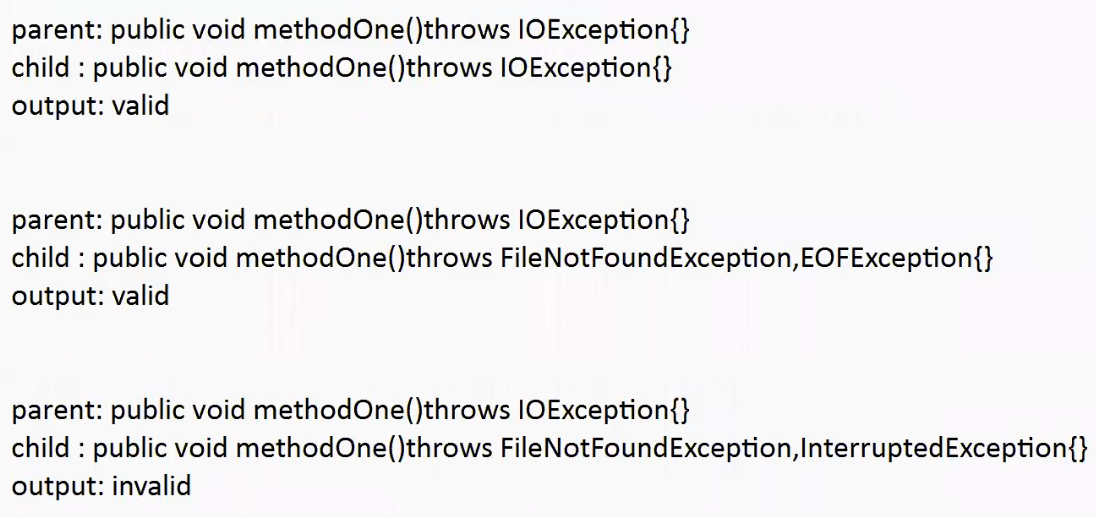
( this rule is applicable from child to parent not from parent to child ( throws checked exception written in parent not in child )

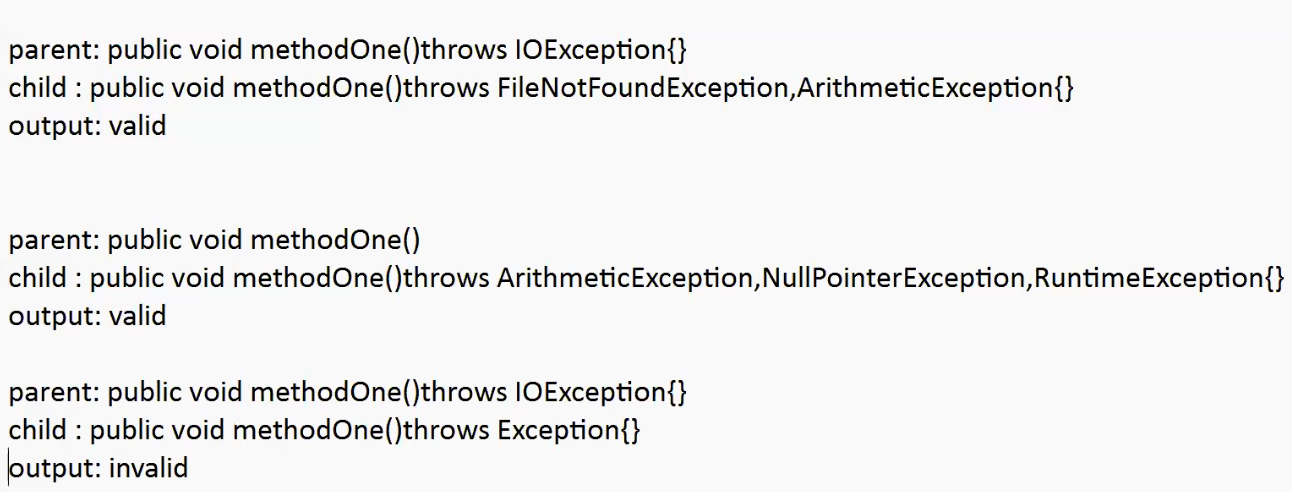
There are no restrictions on Unchecked Exceptions

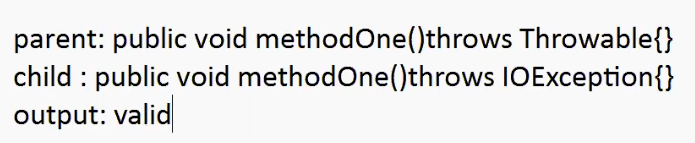


// learn exceptional hierarchy perfectly before going through this concept.









Instanceof

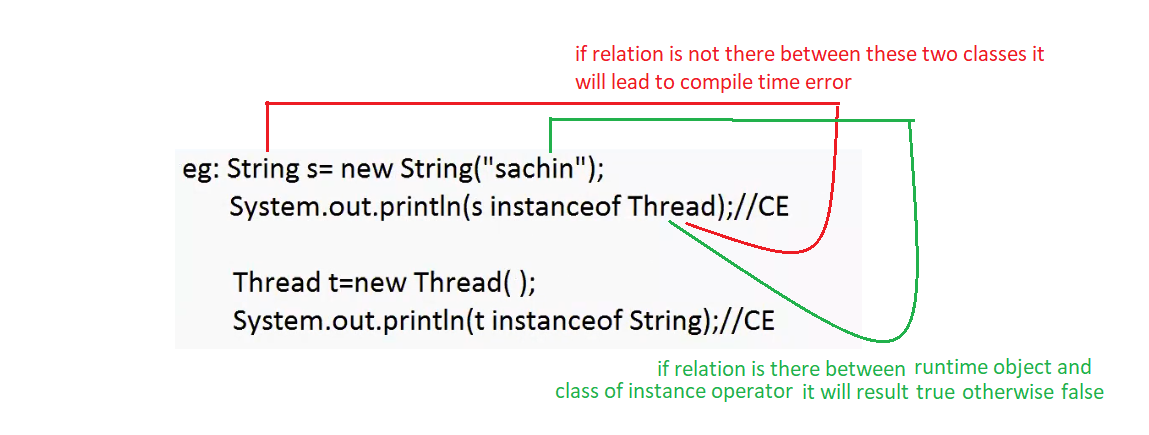
1. we can use instanceof operator to check whether the given object is particular type or not.

r instanceof X

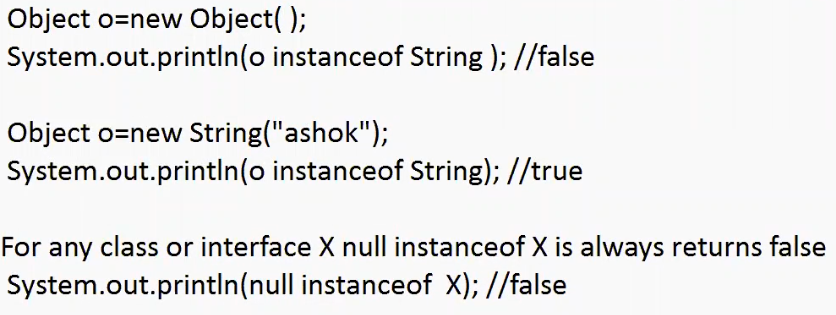
r=> reference

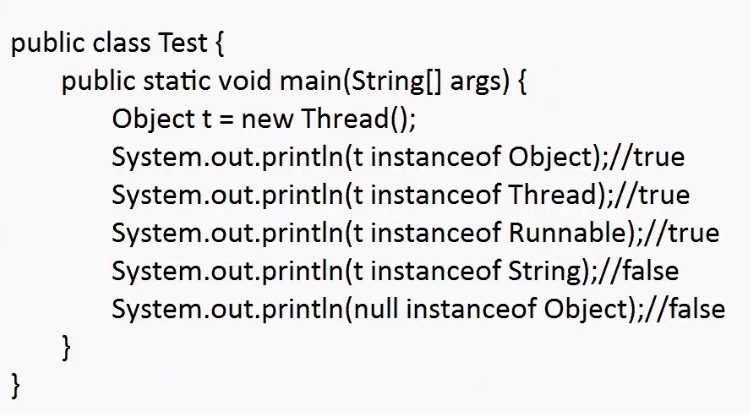
X=> class / interface name

To use instanceof operator compulsory there should be some relation between argument types ( either child to parent (or) parent to child ( or ) same type ) otherwise it will lead to compile time error saying incompatible types.



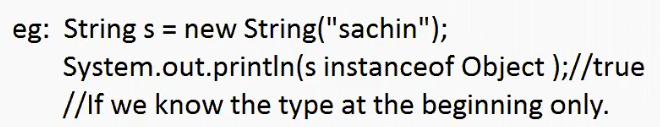
If there is a parent and child relation between runtime object and object of instance operator it will result in false





Difference between instanceof and isIntance() :

Instanceof operator which can be used to check whether the given object is particular type or not . ( it is used when we know the type of object at during compile time )



isInstance()

isInstance() is a method present in class Class. We can use isInstance() method to check whether the given object is particular type or not

( it is used when we know the type during the runtime dynamically)

