**OOPs Concepts**

**Data Hiding**

Outside person can't access our internal data directly or our internal data should not go out directly this oop feature is nothing but data hiding

By declaring data member (variables) as private we can achieve data hiding

the main advantage of data hiding is security

**\*\*\*\*Note** : it is highly recommended to declare data member(variable) as private

**Abstraction**

Hiding internal implementation and just highlight the set of services what we are offering is the concept of abstraction.

the main advantages of abstraction are

1. we can achieve security because we are not highlighting our internal implementation.

2. without affecting outside person we can able to perform any type of changes in our internal system and hence enhancement will become easy.

3. it improves maintainability of the application.

4. it improves easiness to use our system.

5. by using interfaces and abstract classes we can implement abstraction.

**Encapsulation**

The process of binding data and corresponding methods into a single unit is nothing but encapsulation

encapsulation = datahiding + abstraction;

The main advantages of encapsulation are

1. security

2. enhancement become easy : it improves maintainability of the application

\*\*\*\*\*\*\*\*The main advantage of encapsulation is we can achieve security but the main disadvantage of encapsulation is it increase length of the code and slows down execution.

**Tightly encapsulated class**

A class is said to be tightly encapsulated if and only if each and every variable declared as private.

\*\*\*whether class contains corresponding getter and setter methods are not and whether these methods are declared as public or not these things we are not required to check.

**\*\*\*Note:** if the parent class is not tightly encapsulated then no child class is tightly encapsulated

**Is-A relationship**

1. It is also known as inheritance.

2. The main advantage of is-a relationship is code re-usability.

3. By using extends keyword we can implement is-a relationship.

**\*\*\*\*Note :** the most common methods which applicable for any type of child, we have to define in parent class.

🡪the specific methods which are applicable for a particular child we have to define in child class.

* Total java API is implemented based on inheritance concept
* The most common methods which are applicable for any java object are defined in object class. and hence every class in java is the child class of object either directly or indirectly so that object class methods by default available to every java class without rewriting due to this Object class access root for all java classes.
* Throwable class defines the most common methods which are required for every exception and error classes hence this class acts as root for java exception hierarchy.

**Multiple Inheritance**

if our class doesn't extend any other class then only our class is direct child class of object

if our class extends other class then our class is indirect child class of object.

child of child of

B----------------->A----------------->Object

**\*\*\*\*\*Note** : either directly or indirectly java won't provide support for multiple inheritance with respect to classes.

**java provides support for multiple inheritance with respect to interfaces.**

Even though multiple method declarations are available but implementation is unique and hence there is no chance of ambiguity problem in interfaces.

**\*\*\*\*Note** : strictly speaking throw interfaces we won't get any inheritance.

cyclic inheritance is not allowed in java off-course it's not required

**Has-a relationship:**

1. Has a relationship is also known as composition or aggrigation.

2. there is no specific keyword to implement has a relation but most of the times we are depending on new keyword.

3. the main advantage of has a relationship is reusability of the code.

class Car{

Engine e = new Engine();

}

Car has a engine reference.

**Composition vs aggregation**

Without existing container object if there is no chance of existing contained objects, then container and contained objects are strongly associated and this strong association is nothing but **composition.**

e.g : university consist of several departments without existing university there is no chance of existing department hence university and department are stronly associated and this strong association is nothing but composition.

Without existing container object if there is chance of existing contained object then container and contained objects are weakly associated and this weak association is nothing but **aggregation.**

e.g : Department consiste several professors without existing department there may be a chance of existing professor objects hence department and professor objects are weakly associated and this weak association is nothing but aggrigation.

**Note :**

1. in composition objects are strongly associated where as in aggrigation objects are weakly associated.

2. In composition container object holds directly contained objects where as in aggrigation container object holds just references of contained objects.

**Is-A vs Has-A**

1. **if we want total functionality of a class automatically then we should go for is-a relationship**

e.g: complete functionality of Person class required for student class

2. **if we want part of the functionality then we should go for has-a relationship.**

e.g test class contains 100 methods but within demo class I required only 1 or 2 methods then go for has-A relationship.

**Method signature:**

in java method signature consist of method name followed by argument types.

e.g: public static int m1(int i, float f)

m1(int,float) is method signature

return type is not part of method signature in java

compiler will use method signature to resolve method calls

**overloading**

Two method are said to be overloaded if and only if both methods having same name but different argument types

having overloading concept in java reduces complexity of programming

[ in c: to find abs value--- abs(int i), fabs(float f), labs(long l)

In java: abs(int i), abs(float f), abs(long l)]

**\*\*\*\*in overloading method resolution always takes care by compiler based on reference type hence overloading is also considered as compile time polymorphism or static polymorphism or early binding (this is why we can call only parents method with parent reference: P p=new C())**

**Case 1: Automatic promotion in overloading**

1. while resolving overloaded methods if exact match is not available then we won't get any compile time error immediately first it will promote argument to the next level and check whether matched method is available or not.

2. if matched method is available then it will be considered and if matched method is not available then compiler promotes argument once again to the next level. This process will be continued until all possible promotions, still if matched method is not available then we will get compile time error.

Like byte--->short--->int--->long--->float--->double and char--->int

3. while resolving overloaded methods compiler will always gives preference for child type argument when compared with parent type argument

Eg: if we have method with same name has str and obj arguments , and if we pass null then method with str argument is executed

4. m1(string) , m1(stringBuffer) : if I call m1(null) then we will get CE because both arguments classes have same level(both have object has parent class)

5. m1(int i) , m1(int… i) : if I call m1(10) then m1 with integer parameter is executed

**Note** : in general var-arg method will get least priority i.e. if no other method matched then only var-arg method will get the chance. it is exactly same as default case inside switch.

6. **Note** : in overloading method resolution always takes care by compiler based on reference type. In overloading runtime object won't play any role .

**Overriding:**

Whatever method parent has by default available to the child through inheritance, if child class not satisfied with parent class implementation then child is allowed to redefine that method based on its requirement this process is called overriding

the parent class method which is overridden is called overridden method and child class method which is overriding is called overriding method.

\*\*\*\*\***Note: in overriding method resolution is always takes care by JVM based on runtime object and hence overriding is also considered as runtime polymorphism or dynamic polymorphism or late binding.**

**Rules for overriding :**

1. in overriding method names and argument types must be matched. i.e. method signatures must be same

2. in overriding return types must be same but this rule is applicable untile 1.4 version only .

🡪from 1.5 version onwards **we can take co-varient return types according to this, child class method return type need not be same as parent method return type, it's child type also allowed.**

3. Co-varient return type concept applicable only for object types but not for premitive types

4. parent class private methods not available to the child and hence overriding concept not applicable for private methods.

5. Based on our requirement we can define exactly same private method in child class it is valid but not overriding.

6. we can't override parent class final method in child classes if we are trying to override we will get compile time error.

7. parent class abstract method we should override in child class to provide implementation.

8. we can override non abstract method as abstract

the main Advantage of this approach is we can stop availability of parent method implementation in the next level child classes.

Abstract🡨🡪Non-abstract : same applicable to synchronized, native, strictfp modifiers

9. while overriding we can't reduce scope of access modifier but we can increase the scope.

10. Exception in overriding: if child class method throw any checked exception compulsory parent class method should throw the same checked exception or it's parent, otherwise we will get compile time error but there are no restrictions for unchecked exceptions.

11. Static method overriding: we can't override a static method as non-static otherwise we will get compile time error.

Similarly we can't override as non static method as static

If both parent class and child class methods are static then we won't get any compile time error it seems overriding concept applicable for static methods but it's not overriding and it is **method hiding.**

12. overriding with respect to variables: overriding concept applicable only for methods but not for variables



**\*\*\*NOTE** : in overloading we have to check only method names (must be same) and argument types (must be different) we are not required to check remaining like return types access modifiers etc.

But in overriding every thing we have to check like method name, argument types, return types access modifiers, throws class etc.

**Polymorphism:**

one name but multiple forms is concept of polymorphism.

Overloading and overriding is example of polymorphism

usage of parent reference to hold child object is concept of polymorphism or is also considered as polymorphism

**\*\*\*Note: When we should go for parent reference to hold child object.**

if we don't know exact return type of object then we should go for parent reference.

e.g the first element present in the array list can be any type. it may be student object, or customer, String, stringBuffer object hence the return type of get method is object, which can hold any object.

**3 Pillars of OOPS**

**Encapsulation-------security**

**Polymorphism------Flexibility**

**Inheritence----------Re-usability**

**Coupling**

The degree of dependency of components is called **coupling**. if dependency is more then it is considered as **tightly coupling** and if dependency is less then it is considered as **loosly coupling.**

Tightly coupling is not recommended because enhancement will be difficult as there will be more dependency, supress reusability, reduces maintainability

**Cohesion**

For every component **a clear well defined functionality** is defined then that component is said to be follow high cohesion

**\*\*\*\*Note : loosely coupling and high cohesion are good programming practices**

**Object typecasting**

we can use parent reference to hold child object.

we can use interface reference to hold implemented class object.

**A b = (C) d;**

Class| name of class|interface ref var name

Interface ref var

**Mantra 1:(compile time checking 1)**

The type of 'd' and 'c' must have some relation either child to parent or parent to child or same type otherwise we will get compile time error saying inconvertable types found 'd' type required 'c'.

**Mantra 2:(compile time checking 2)**

'C' must be either same or derived type of 'A' otherwise we will get compile time error saying incompatible types found : C required : A

**Mantra 3 : (runtime checking)**

Runtime object type of 'd' must be either same or derived type of 'c' otherwise we will get runtime exception saying ClassCastException

**\*\*\*Note:** Strictly speaking throw type casting we are not creating any new object.

For the existing object we are providing another type of reference variable i.e we are performing type casting but not object casting.

In overriding, method resolution is always based on runtime object

In method hiding(static) , method resolution is always based on reference type

Variable resolution is always based on runtime object

**Static control flow**

Whenever we are executing a java class the following sequence of activities or steps will be executed as the part of static control flow.

1. Identification of static members from top to bottom.

2. execution of static variable assignments and static blocks from top to bottom.

3. execution of main method.

--------------------------------------

RIWO : READ INDIRECTLY WRITE ONLY

inside static block if we are trying to read a variable that read operation is called direct read.

if we are calling a method and within that method if we are trying to read a variable that read operation is called indirect read.

if a variable is just identified by the jvm and original value not yet assigned then the variable is said to be Read Indirectly Write Only State[RIWO].

If a variable is in read indirectly write-only state then we can't perform direct read but we can perform indirect read.

If we are trying to read directly then we will get compile time error saying IllegalForwardReference.

**\*\*\*\*\*Static block** will be executed at the time of class loading hence at the time of class loading if we want to perform any activity we have to define that inside static block.

within a class we can declare any number of static blocks but all these static blocks will be executed from top to bottom.

**Static control flow in parent to child relationship**

Whenever we are executing child class the following sequence of events will be executed automatically as part of static control flow.

1. Identification of static members from parent to child.

2. Execution of static variable assignments and static blocks from parent to child.

3. Execution of only child class main method.

**\*\*\*\*\*\*NOTE:** whenever we are loading child class automatically parent class will be loaded. But whenever we are loading parent class child class won't be loaded (because parent class members by default available to the child class where as child class members by default won't available to the parent).

**Instance control flow**

1. identification of instance members from top to bottom.

2. execution of instance variable assignments and instance blocks from top to bottom.

3. execution of constructor.

**\*\*\*\*NOTE:**

* static control flow is one time activity which will be performed at the time of class loading.
* But instance control flow is not one time activity and it will be performed for every object creation.
* object creation is most costly operation if there is no specific requirement then it is not recommended to create object.

**Instance control flow in parent to child relationship**

1. identification of instance members from parent to child.

2. Execution of instance variable assignments and instance blocks only in parent class

3. Execution of parent constructor.

4. Execution of instance variable assignments and instance blocks in child class.

5. Execution of child constructor

**\*\*NOTE:** From static area we can't access instance members directly because while executing static area JVM may not identify instance members.

**WAYS TO CRATE AN OBJECT IN JAVA OR WAYS TO GET OBJECT IN JAVA.**

1. By using new operation

Test t = new Test();

2. By using new instance method

Test t = (Test) Class.forName("Test")newInstance();

3. By using factory methods

Runtime r = Runtime.getRuntime();

e.g dateFormat df = DateFormat.getInstance();

4. By using clone() method.

Test t1 = new Test();

Test t2 = (Test)t1.clone();

5. By using deserilalization

FileInputStream fis = new FileInputStream("abc.sar");

ObjectInputStream ois = new ObjectInputStream(fis);

Dog d2 = (Dog)ois.readObject();

**Constructor**

the main purpose of constructor is to perform initialization of an object

**\*\*\*\*Note:** other than initialization if we want to perform any activity for every object creation then we should go for instance block

**Rules of writing constructors:**

1. Name of the class and name of the constructor must be matched.

2. Return type concept is not applicable for constructor even void also.

3. By mistake if we are trying to declare return type for the constructor then we won't get any compile-time error because compiler treats it as a method.

**\*\*\*\*Note:**

* The only applicable modifiers for constructors are public, private, protected, default if we are trying to use any other modifier we will get compile time error.
* Compiler is responsible to generate default constructor but not JVM
* if we are not writting any constructor then only compiler will generate default constructor i.e. if we are writting atlist one constructor compiler won't generate default constructor hence every class in java can contain constructor it may be default constructor generated by compiler or costumized constructor explicitly provided by compiler but not both symeltuniously.

**Protoype of default constructor**

1. It is always no-arg constructor

2. the access modifier of default constructor is exactly same as access modifier of class (this rule is applicable only for public and default)

3. it contains only one line super();

**\*\*\*\*Note:** The first line inside every constructor should be either super or this.

and if we are not writting any thing then compiler will always place super().

we can call a constructor directly from another constructor only.

**super or this**

* we can use only in first line
* we can use only one but not both symulteniously
* we can use only in constructor

super(), this() super, this

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. These are constructor calls These are keywords

to call super class and current to referrer super class and

class constructors current class instance members

2. we can use only in constructors we can use anywhere except static area

as first line

3. we can use only once in constructor we can use any number of times.

**Overloaded constructors**

within a class we can declare multiple constructors and all these constructors having same name but different type of arguments. Hence all these constructors are considered as **overloaded constructors**. Hence overloading concept applicable for constructors.

**\*\*\*\*Note:**

* for constructors inheritance and overridding concepts are not applicable but overloading concept is applicable
* every class in java including abstract class can contains constructor but interface doesn't contain constructor.

**\*\*\*Case 1:** recursive method call is a runtime exception saying stackOverFlowError

but in our program if there is a chance of recursive constructor invocation then the code won't compile and we will get compile time error.

**\*\*\*Case 2:**

1. if parent class contains any argument constructor then while writting child classes we have to take special care with respect to constructors.

2. whenever we are writting any argument constructor it is highly recommanded to write no-arg constructor also.

**\*\*\*Case 3:**

if parent class throws any checked exception compulsary child class constructor should throw same checked exception or its parent. otherwise the code won't compile.

**Singleton classes**

for any java class if we are allowed to create only one object such type of class is called singleton class.

e.g Runtime, BusinessDeligate, ServiceLocator

**Advantage of singletone class :**

if several people have same requirement then it is not recommanded to create seprate object for every requirement.

We have to create only one object and we can reuse same object for every similar requirement so that performance and memory utilizations will be improved.

**How to create our own singleton classes?**

we can create our own singleton classes for this we have to use private constructor and private static variable and public factory method.

|  |  |
| --- | --- |
| class Test{  private static Test testSingleton = new Test();  private Test(){  }  public static Test getTest(){  return testSingleton;  }  } | class Test{  private static Test testSingleton;  private Test(){  }  public static Test getTest(){  if(testSingleton == null){  testSingleton = new Test();  }  return testSingleton;  }  } |

**\*\*\*\*NOTE:**

At any point of time for test class we can create only one object hence test class is singleton class.

class is not final but we are not allowed to create child classes How it is possible.

by declaring every constructor as private we can restrict child class creation.

**Exception:**

An unexpected unwanted event that disturbs the normal flow of the program is called exception.

It is highly recommended to handle exceptions, and the main objective of exception handling is **graceful termination** of the program.

Exception handling doesn't mean repairing an exception we have to **provide alternative way** to continue rest of the program normally is the concept of exception handling.

try{

read data from remote file locating at London

} catch(FileNotFoundException e){

use local file and continue rest of the program normally.

}

**\*\*\*Note: database example:** open DB connection ------ read data --------close DB connection

If something goes wrong during read then also we should have a way to close DB connection

**RuntimeStack mechanism:**

For every thread JVM will create a runtime stack, each and every method calls performed by that thread will be stored in the corresponding stack.

each entry in the stack is called stack frame or activation record

after completing every method call the corresponding entry from the stack will be removed.

After completing all method calls the stack will become empty and that empty stack will be destroyed by JVM just before terminating the thread.

**Default Exception handling in java**

inside a method if any exception occurred, the method in which it is raised is responsible to create exception object by including the following information

1. Name of Exception
2. Description of exception
3. Location of at which exception occurs. (stack trace)

2. After creating exception method handover that object to the JVM

3. JVM will check whether the method contains any exception handling code or not, if the method doesn't contain exception handling code then JVM terminates that method abnormally and removes the corresponding entry from the stack then JVM identifies identifier and checks whether caller method contain any handling code or not if the caller method doesn't contain handling code then JVM terminates the caller method also abnormally and removes corresponding entry from stack.

4. This process will be continued until main method and if the main() method also doesn't contain handling code then JVM terminates main() method also abnormally and removes corresponding entry from stack.

5. Then JVM handover responsibility of exception handling to default exception handler, which is the part of JVM. Default exception handler prints exception information in the **following format** and terminates program abnormally.

**Exception in thread "XXX" name of the Exception : Description stackTrace.**

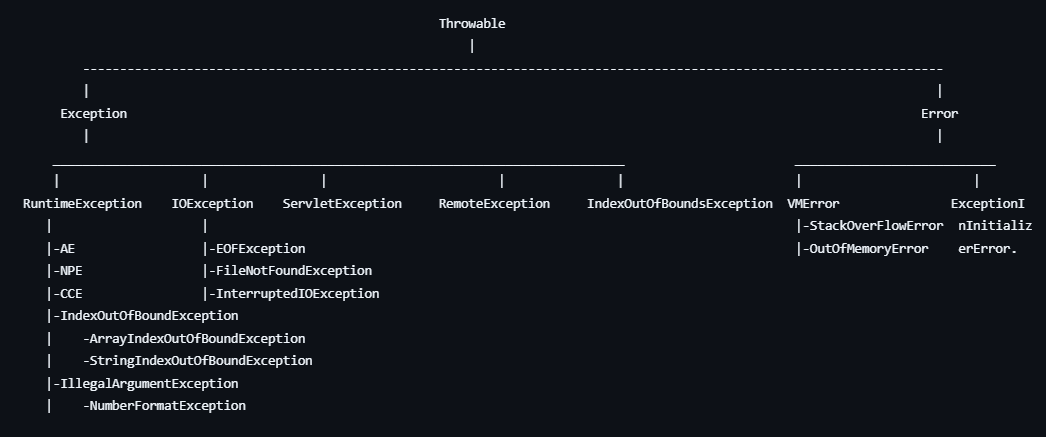
**Exception hierarchy**

Throwable class acts as root for java exception hierarchy.

Throwable class defines two child classes

1. **Exception** : Most of the times exceptions are caused by our program and these are recoverable.

2. **Error :** Most of the times errors are not caused by our program and these are due to lack of system resources Errors are non recoverable



**Checked Exceptions vs unchecked Exceptions**

1. Exceptions which are checked by compiler for smooth execution of the program are called **checked exceptions**

2. In our program if there is a chance of rising checked exception then compulsory we should handle that checked exception(either by try catch or by throws keyword.) otherwise we will get compile time error.

3. The exceptions which are not checked by compiler whether programmer handling or not such type of exceptions are called **unchecked exceptions.**

**\*\*\*Note:**

1. Whether it is checked or unchecked every exception occurs at run-time only there is no chance of occurring any exception at compile time.

2. RuntimeException and its child classes, error and its child classes are unchecked. except these remaining are checked.

3. **FullyChecked vs Partially Checked.**

4. A checked exception is said to be **fully checked** if and only if all its child classes also checked [e.g IOException, InterruptedException]

5. A checked exception is said to be **partially checked** if and only if some of its child classes are unchecked. [e.g Exception, Throwable]

**The only possible partially checked exceptions in java are**

**1. Exception**

**2. Throwable**

**Try-Catch**

try{

Risky code

}catch(Exception e){

handling code.

}

**Control flow in try-catch**

**\*\*\*Note** : Within the try block if anywhere exception rised then rest of the try block won't be executed even though we handled that exception. Hence within the try block we have to take only risky code and length of try block should be as less as possible.

In addition to try block there may be a chance of rising the exception inside catch and finally blocks.

if any statement which is not part of try block and rises an exception then it is always abnormal termination.

**Methods To Print Exception Information**

method printable format

---------------------------------------------------------------------------------------------------

1**. printStackTract**() : Name of exception : Description StackTrace.

2. **toString**() : Name of Exception : Description

3. **getMessage**() : Description.

try{

sopln(10/0);

}catch(Exception e){

e.printStackTrace();---------> java.lang.ArithmaticException : / by zero at Test main(); sopln(e.toString());---------> java.lang.ArithmaticException : / by zero  
sopln(e.getMessage);---------> / by zero ;

}

internally **default exception handle**r will use printStackTract() method to print exception information to the console.

**Try with multiple catch block**

it is highly recommended to take separate catch block i.e try with multiple catch block is always possible and recommended to use.

**\*\*\*Note:** if try with multiple catch blocks present then the order of catch blocks is very important we have to take child first and then parent otherwise we will get compile time error saying

we can't declare two catch blocks for the same exception otherwise we will get compile time error.

Good programming practice recommended to use:

try{

……

}catch(ArithmaticException e){

perform alternative arithmetic operation

}catch(SQLException e){

use mysql db instead of oracle db

}catch(FileNotFoundException e){

use local file instead of remote file

}catch(Exception e){

default exception handling

}

**Finally**

finally is a block always associated with try-catch to maintain cleanup code

**\*\*\*Note:** the specialty of finally block is it will be executed always irrespective of whether exception is raised or not raised and whether handled or not handled.

**finallize()**

finalize is a method always invoked by garbage collector just before destroying an object to perform cleanup activities once finalize method completes immediately garbage collector destroys that object.

**\*\*\*Note:**

* **finally block** is responsible to perform cleanup activities related to try block i.e whatever resources we opened as a part of try block will be closed inside finally block.
* where as **finalize method** is responsible to perform cleanup activities related to object i.e whatever resources associated with object will be deallocated before destroying an object by using finalize method.

**various possible combinations of try-catch-finally.**

1. in try-catch-finally order is important

2. whenever we are writing try compulsory we should write either catch or finally otherwise we will get compile time error i.e try without catch or finally is invalid.

3. whenever we are writing catch block compulsory try must be required i.e catch without try is invalid

4. whenever we are writing finally block compulsory we should write try block i.e finally without try is invalid.

5. inside try catch and finally blocks we can declare try-catch-finally blocks i.e nesting of try-catch-finally is allowed.

6. for try-catch-finally blocks curly braces are mandatory.

**Throw keyword**

Programmer throws Exception object and JVM will catch Exception object

Some times we can create exception object explicitly we can handover to the JVM manually for this we have to use throw keyword

Hence the main objective of throw keyword is to handover our created exception object to the JVM manually

If a exception occurs in method then that method is responsible to create exception object and handover to JVM internally but using throw programmer manually creates and handover it to JVM manually

**\*\*\*Note** : Best use of throw keyword is for user defined exceptions or customized exceptions.

**1. throw e;** if e refers null then we will get NullPointerException.

2. After throw statement we are not allowed to write any statement directly otherwise we will get compile time error saying unreachable

3. we can use throw keyword only for Throwable types if we are trying to use for normal java objects we will get compile time error saying incomputable types.

**Throws keyword**

in our program if there is possibility of rising checked exception then compulsory we should handle that checked exception otherwise we will get compile time error saying :

Unreported exception XXX must be caught or declared to be thrown.

we can handle this compile time error by using the following **two ways**

1. **by using try-catch.**

2. **By using throws keyword**: we can use throws keyword to deligate responsibility of exception handling to the caller(it may be another method or JVM) then caller method is responsible to handle that exception.

throws keyword required only for checked exceptions, usage of throws keyword for unchecked exceptions there is no use or there is no impact.

throws keyword required only to convince compiler and usage of throws keyword doesn't prevent abnormal termination of the program.

class Test{

public static void main(String[] args){

doStuff();

}

public static void doStuff(){-------------------------->CE : unreported exception j.l.IE

doMoreStuff(); must be caught or declared to

} be thrown.

public static void doMoreStuff() throws IE{

Thread.sleep(10000);

}

}

**it is recommended to use try-catch over throws keyword**

**\*\*\*\*Note:**

we can use throws keyword for methods and constructors but not for classes.

we can use throws keyword only for Throwable types if we are trying to use for normal java classes then we will get compile time error saying incompatible types.

within the try block if there is no chance of rising an exception then we can't write catch block for that exception otherwise we will get compile time error saying Exception XXX is never thrown in body of corresponding try statement but this rule is applicable only for fully checked exceptions.

**Exception handling keywords summery**

1. try : to maintain risky code

2. catch : to maintain exception handling code.

3. finally : to maintain cleanup code.

4. throw : to handover our created exception object to the JVM manually.

5. throws : to deli-get responsibility of exception handling to the caller.

**Various possible compile-time errors in exception handling**

1. unreported exception XXX must be caught or declared to be thrown.

2. Exception XXX has already been caught

3. Exception XXX is never thrown in body of corresponding try statement.

4. Unreachable statement.

5. Incompatible types found test required java.lang.throwable

6. Try without catch or finally.

7. catch without try

8. finally without try.

**Customized or user defined exceptions**

Some times to meet programming requirements we can define our own exceptions such type of exceptions are called customized or user defined exceptions

class TooYoungException extends RuntimeException{

TooYoungException(String s){

super(s); ------------------------> **To make description available to default exception handler.**

}

}

**\*\*Note:**

1. throw keyword is best suitable for user defined or customized exceptions but not for pre-defined exceptions.

2. It is highly recommended to define customized exceptions as unchecked i.e we have to extends RuntimeException class but not Exception.

**Top 10 Exceptions**

Based on the person who is rising an exception all exceptions are divided into two categories.

**1. JVM Exceptions**

**2. Programatic Exceptions.**

The exceptions which are raised automatically by JVM whenever a particular event occurs are called **JVM Exceptions.**

the exceptions which are raised explicitly either by programmer or by API developer to indicate that some thing goes wrong are called **programmatic exceptions.**

**1.ArrayIndexOutOfBoundsException**: child class of runtime exception so it is unchecked

**2. NullPointerException**: child class of runtime exception so it is unchecked

**3. ClassCastException**: child class of runtime and uncheked(raised when we try to typecast parenr object to child object)

**4. StackOverFlowError**: child class of Error and unchecked (raised during recursive method call)

**5. NoClassDefFoundError**: child class of Error and unchecked(raised when **JVM** unable to find required .class file.)

**6. ExceptionInInitializerError**: child class of Error and unchecked(Raised automatically by **JVM** whenever if any exception occurs while executing static variable assignments and static blocks.)

**7. IllegalArgumentException** : child class of runtime Exception and unchecked( Raised explicitly either by **programmer or by API Developer** to indicate that a method has been invoked with illegalArgument.)

**8. NumberFormatException**: It is the direct child class of IllegalArgumentException which is child class of RunTimeException and hence it is unchecked.(Raised explicitly either by **programmer or by API developer** to indicate that we are trying to convert String to Number and the String is not properly formatted)

Eg: int i = Integer.parseInt("ten"); //RE : NumberFormatException.

**9. IllegalStateException:** It is child class of RuntimeException and hence it is unchecked raised explicitly either by **programmer or by API developer** to indicate that a method has been invoked at wrong time. [ eg: if we try to invoke start method of thread twice , we will get this exception]

**10. AssertionError**: It is the child class of error and hence it is unchecked( raised explicitly by the **programmer or by API developer** to indicate that AssertStatement fails.)

e.g assert(x > 10);

if x is not greater than 10 then we will get RuntimeException saying assertion error.

**1.7 version enhancements with respect to exception handling.**

**1. try with resources**

**2. multi-catch block**

**1. try with resources**

the main advantage of **try with resources** is whatever resources we open as a part of try block will be closed automatically once control reaches end of try block either normally or abnormally and hence we are not required to close explicitly so that complexity of programming will be reduced. we are not required to write finally block so that length of the code will be reduced and readability will be improved.

try(BufferedReader br = new BufferedReader(new FileReader("Input.txt"))){

//use br based on our requirement

//br will be closed automatically once control reaches end of try block either normally or abnormally.

}catch(IOException e){

handling code

}

**\*\*\*Note:**

1. we can declare **multiple resources** but these resources should be **separated with semicolon (';')**
2. All resources should be auto-closable resources
3. A resource is said to be auto-closable if and only if corresponding class implements java.lang.AutoClosable interface(contains only one method: pubic void close()), all io related resources database related resources and network related resources are already implemented auto-closable inerface. Being a programmer we are not required to do any thing just we should aware the point.
4. All resource reference variables are implicitly final and hence within the try block we can't perform re-assignment otherwise we will get compile time error.
5. from 1.7 version we can take only try with resource without catch or finally.

**2. multi-catch block**

According to this we can write a single catch block that can handle multiple different type of exceptions.

the main advantage of this approach is length of the code will be reduced and readability will be improved.

**\*\*\*\*Note:** in multi-catch block there should not be any relation between exception types (either child to parent or parent to child or same type) otherwise we will get compile time error.

**Exception propagation:** Inside a method if an exception raised and if we are not handling that exception then exception object will be propagated to caller then caller method is responsible to handle exception this process is called exception propagation.

**Re-throwing exception:** we can use this approach to convert one exception type to another exception type

e.g try{

sopln(10/0);

}catch(ArithmaticException e){

throw new NullPointerException();

}