**XE:Exception**

1. **Difference between throw and throws?**
   * 1. It includes:
   * Throw is used to trigger an exception where as throws is used in declaration of exception. The throws keyword appears at the end of a method's signature.
   * Without throws, Checked exception cannot be handled where as checked exception can be propagated with throws.

**Throws:**

public void myMethod() throws PRException

{This means the super function calling the function should be equipped to handle this exception.

public void Callee()

{

try{

myMethod();

}catch(PRException ex)

{

...handle Exception....

}

}

}

**Throw:**

try{

if(age>100)

{ throw new AgeBarException(); //Customized ExceptioN

}else{

....

}

}

}catch(AgeBarException ex){

...handle Exception.....

}

1. **Can try statements be nested?**
2. Yes

**class** NestTry {

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

**try** {

**int** a = args.length;

**int** b = 42 / a;

System.out.println("a = " + a);

**try** {

**if** (a == 1)

a = a / (a - a); // division by zero

**if** (a == 2) {

**int** c[] = { 1 };

c[42] = 99; // generate an out-of-bounds exception

}

} **catch** (ArrayIndexOutOfBoundsException e) {

System.out.println("Array index out-of-bounds: " + e);

}

} **catch** (ArithmeticException e) {

System.out.println("Divide by 0: " + e);

}

}

}

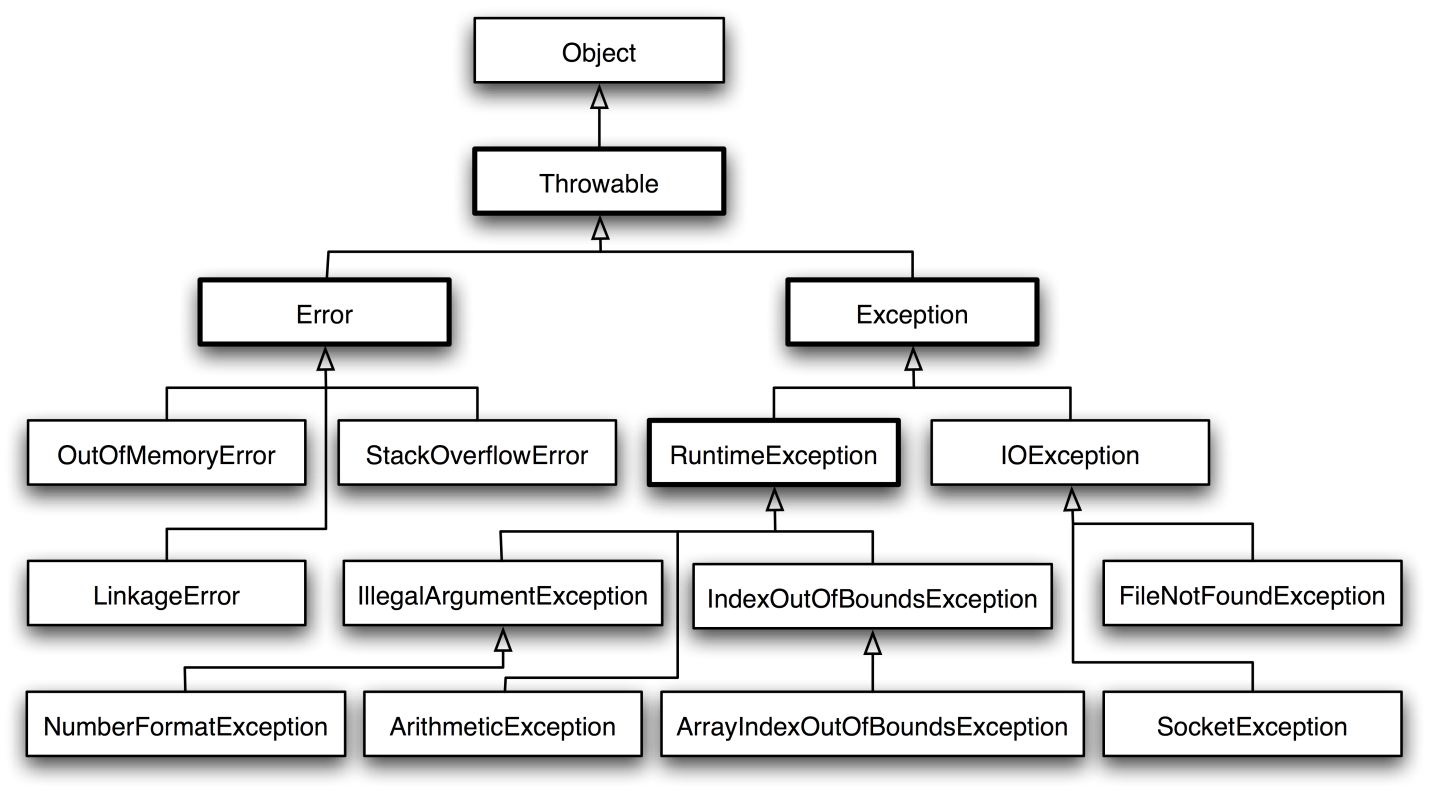
1. **What is an Exception?**
2. An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object, called an exception object, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called throwing an exception.

After a method throws an exception, the runtime system attempts to find something to handle it. The set of possible "somethings" to handle the exception is the ordered list of methods that had been called to get to the method where the error occurred.

Exceptions come in two flavors: checked and unchecked.

* Checked exceptions include all subtypes of Exception, excluding classes that extend RuntimeException.
* Checked exceptions are subject to the handle or declare rule; any method that might throw a checked exception (including methods that invoke methods that can throw a checked exception) must either declare the exception using throws, or handle the exception with an appropriate try/catch.
* Subtypes of Error or RuntimeException are unchecked, so the compiler doesn't enforce the handle or declare rule. You're free to handle them, or to declare them, but the compiler doesn't care one way or the other.

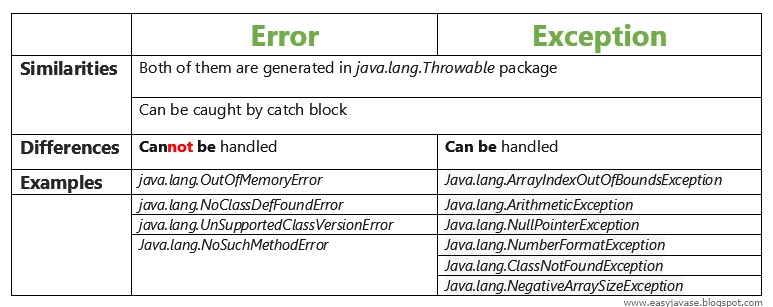
****

Important points for exceptions:

* If you use an optional finally block, it will always be invoked, regardless of whether an exception in the corresponding try is thrown or not, and regardless of whether a thrown exception is caught or not.
* The only exception to the finally-will-always-be-called rule is that a finally will not be invoked if the JVM shuts down. That could happen if code from the try or catch blocks calls **System.exit().**
* Just because finally is invoked does not mean it will complete. Code in the finally block could itself raise an exception or issue a System.exit().
* Uncaught exceptions propagate back through the call stack, starting from the method where the exception is thrown and ending with either the first method that has a corresponding catch for that exception type or a JVM shutdown (which happens if the exception gets to main(), and main() is "ducking" the exception by declaring it).
* You can create your own exceptions, normally by extending Exception or one of its subtypes. Your exception will then be considered a checked exception, and the compiler will enforce the handle or declare rule for that exception.
* **All catch blocks must be ordered from most specific to most general**. If you have a catch clause for both IOException and Exception, you must put the catch for IOException first in your code. Otherwise, the IOException would be caught by catch(Exception e), because a catch argument can catch the specified exception or any of its subtypes! **The compiler will stop you from defining catch clauses that can never be reached.**
* Some exceptions are created by programmers, some by the JVM.

1. **What is the difference between error and an exception?**
2. An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. Exceptions are conditions that occur because of bad input etc. e.g. FileNotFoundException will be thrown if the specified file does not exist.

Subtypes of Error or RuntimeException are unchecked, so the compiler doesn't enforce the handle or declare rule. You're free to handle them, or to declare them, but the compiler doesn't care one way or the other.

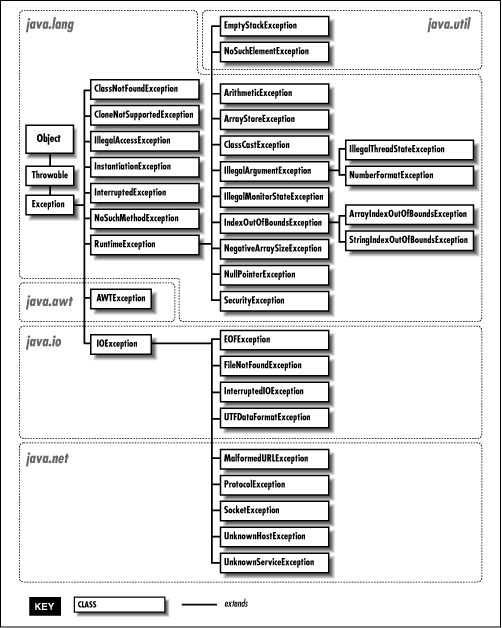


1. **Is it necessary that each try block must be followed by a catch block?**
2. It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block or a finally block.
3. **Explain Runtime Exceptions?**
4. It is an exception that occurs that probably could have been avoided by the programmer. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation.
5. **Which are the two subclasses under Exception class?**
6. The Exception class has two main subclasses : IOException class and RuntimeException Class.
7. **What is NullPointerException?**

A:  A NullPointerException is thrown when calling the instance method of a null object, accessing or modifying the field of a null object etc.

1. **Does it matter in what order catch statements for FileNotFoundException and IOException are written?**

**A:**  Yes, it does. The FileNotFoundException is inherited from the IOException. Exception's subclasses have to be caught first.



1. **When is the ArrayStoreException thrown?**
2. Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects. For example, the following code generates an ArrayStoreException:

Object x[] = new String[3];

x[0] = new Integer(0);

1. **When ArithmeticException is thrown?**
2. The ArithmeticException is thrown when integer is divided by zero or taking the remainder of a number by zero. **It is never thrown in floating-point operations.**

Thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero" throws an instance of this class. ArithmeticException objects may be constructed by the virtual machine as if suppression was disabled and/or the stack trace was not writable.

1. **What class of exceptions is generated by the Java run-time system?**
2. The Java runtime system generates RuntimeException and Error exceptions.
3. **What things should be kept in mind while creating your own exceptions in Java?**
4. While creating your own exception:

* All exceptions must be a child of Throwable.
* If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
* You want to write a runtime exception, you need to extend the RuntimeException class.

How you write your own exception?