Exception Handling

1. A Java Exception is an Object that describes the exception that occurs in a program. When an exceptional event occur in java, an exception is said to be thrown
2. Checked Exception: - The exception that can be predicted by the programmer. Ex- File not found. These type of exceptions must be checked at compile time. Some checked exceptions are: SQLException,IOException,DataAccessException,CloneNotSupportedException,InterruptedException
3. Unchecked Exception: - are the classes that extends RuntimeException. Unchecked Exceptions are ignored at compile time and are checked at runtime. Ex- AirthmeticException, NullPointerException, ArrayIndexOutOfBoundException, IllegalArgumentException
4. Error: - Errors are ignored in code because we cannot do anything about the error. Ex- if stack overflow occurs, an error will arise.
5. Exception handling is done by transferring the execution of a program to an appropriate exception handler when exception occurs
6. Example for Unreachable Catch block: - While using multiple catch statements, it is **important to remember** that exception sub-classes inside **catch** must come **before any of their super classes** otherwise it’ll lead to Compile Time Error

class Excep

{

public static void main(String[] args)

{

try

{

int arr[]={1,2};

arr[2]=3/0;

}

catch(Exception e) //This block handles all Exception

{

System.out.println("Generic exception");

}

catch(ArrayIndexOutOfBoundsException e) //This block is unreachable

{

System.out.println("array index out of bound exception");

}

}

}

1. Nested try block is used when a part of a block may cause one error while entire block may cause another error. In case, if inner **try** block does not have a **catch** handler for a particular exception then the outer **try** is checked for match
2. **Important points to Remember**
3. If you do not explicitly use the try catch blocks in your program, java will provide a default exception handler, which will print the exception details on the terminal, whenever exception occurs.
4. Super class **Throwable** overrides **toString()** function, to display error message in form of string.
5. While using multiple catch block, always make sure that exception subclasses comes before any of their super classes. Else you will get compile time error.
6. In nested try catch, the inner try block, uses its own catch block as well as catch block of the outer try, if required.
7. Only the object of Throwable class or its subclasses can be thrown.
8. Throw keyword: - It is used to explicitly throw exception as well as custom exception. We can throw **either checked or unchecked** exception. Only object of Throwable class or its sub classes can be thrown. Program execution stops on encountering **throw** statement, and the closest catch statement is checked for matching type of exception. After throw statement we cannot write any code below this, otherwise we’ll get CTError UnreachableStatement.
9. Throws keyword: - Throws is mainly used for propagate **checked** exception and used to specify the method level exception. Means, it is used to propagate the checked exception to the caller of the method that use throws keyword. It is used to declare the exception, it provides information to the programmer that there may occur an exception so during call of that method, programmer must use exception handling mechanism.

If a method A() can throw any checked exception, then we declare this by using throws keyword.

A() throws Exception

If we need the method A() anywhere in the program then, while calling this method we need to use try/catch block or using throws keyword to calling method(here main) to handle this exception.

For Example:-

Class MyException extends Exception{ // Exception is checked exception

MyException(String s){

super(s); // Creating custom exception

}

}

Class ex2{

Static void valid(int i) throws MyException{

if(i<18){

throw new MyException(“Not Valid”);

}

}

public static void main(String args[]){

try{

valid(12);

}catch(MyException e){System.out.println(e);}

}

}

1. **Points to remember** when using Method Overriding with Exception Handling
2. If super class method does not declare any exception, then sub class overridden method **cannot** declare **checked exception** but it **can** declare **unchecked exception**

class Super

{

void show() { System.out.println("parent class"); }

}

public class Sub extends Super

{

void show() throws IOException //Compile time error

{ System.out.println("parent class"); }

public static void main( String[] args )

{

Super s=new Sub();

s.show();

}

}

As the method show() doesn’t throws any exception while in Super class, hence its overridden version can also **not throw any checked exception**

import java.io.\*;

class Super

{

void show(){ System.out.println("parent class"); }

}

public class Sub extends Super

{

void show() throws ArrayIndexOutOfBoundsException //Correct

{ System.out.println("child class"); }

public static void main(String[] args)

{

Super s=new Sub();

s.show();

}

}

**Because ArrayIndexOutOfBoundException** is an unchecked exception, hence overrided show() method can throw it.

1. If super class method throws an exception, then subclass overridden method can throw the **same** exception or **no** exception, but **must not throw parent exception** of the exception thrown by Super class method.

It means, if Super class method throws object of NullPointerException class, then Subclass method can either throw same exception or can throw no exception, but it can never throw object of Exception class(Parent of NullPointerException class).

import java.io.\*;

class Super

{

void show() throws ArithmeticException

{ System.out.println("parent class"); }

}

public class Sub extends Super {

void show() throws Exception //Cmpile time Error

{ System.out.println("child class"); }

public static void main(String[] args)

{

try {

Super s=new Sub();

s.show();

}

catch(Exception e){}

}

}

1. List of Common Checked Exceptions in Java

Common checked exceptions defined in the java.lang package:

* ReflectiveOperationException
  + ClassNotFoundException
  + InstantiationException
  + IllegalAccessException
  + InvocationTargetException
  + NoSuchFieldException
  + NoSuchMethodException
* CloneNotSupportedException
* InterruptedException

Common checked exceptions defined in the java.io package:

* IOException
  + EOFException
  + FileNotFoundException
  + InterruptedIOException
  + UnsupportedEncodingException
  + UTFDataFormatException
  + ObjectStreamException
* InvalidClassException
* InvalidObjectException
* NotSerializableException
* StreamCorruptedException
* WriteAbortedException

Common checked exceptions defined in the java.net package (almost are subtypes of IOException):

* SocketException
  + BindException
  + ConnectException
* HttpRetryException
* MalformedURLException
* ProtocolException
* UnknownHostException
* UnknownServiceException

Common checked exceptions defined in the java.sql package:

* SQLException
  + BatchUpdateException
  + SQLClientInfoException
  + SQLNonTransientException
* SQLDataException
* SQLFeatureNotSupportedException
* SQLIntegrityConstraintViolationException
* SQLSyntaxErrorException
  + SQLTransientException
* SQLTimeoutException
* SQLTransactionRollbackException
* SQLTransientConnectionException
  + SQLRecoverableException
  + SQLWarning

1. List of common Unchecked Exception in Java

Common unchecked exceptions in the java.lang package:

* ArithmeticException
* IndexOutOfBoundsException
  + ArrayIndexOutOfBoundsException
  + StringIndexOutOfBoundsException
* ArrayStoreException
* ClassCastException
* EnumConstantNotPresentException
* IllegalArgumentException
  + IllegalThreadStateException
  + NumberFormatException
* IllegalMonitorStateException
* IllegalStateException
* NegativeArraySizeException
* NullPointerException
* SecurityException
* TypeNotPresentException
* UnsupportedOperationException

Common unchecked exceptions in the java.util package:

* ConcurrentModificationException
* EmptyStackException
* NoSuchElementException
  + InputMismatchException
* MissingResourceException

Multi-Threading

* 1. Thread Class: - Thread class is the main class on which Java’s Multithreading system is based. Thread class & Runnable interface is used to create and run threads for utilizing multithreading feature of java
  2. Constructors of Thread class :-
     1. Thread()
     2. Thread(String str)
     3. Thread(Runnable r)
     4. Thread(Runnable r, String str)
  3. Method of Thread Class: -
     1. setName() – to give thread a name
     2. getName() – return thread’s name
     3. getPriority() – return thread’s priority
     4. isAlive() – checks if thread is still running or not
     5. join() –Using this method, we tell our thread to wait until the thread on which it is called completes its execution
     6. join(long milliseconds)- It allows us specific time for which we want to wait for the specified thread to terminate.
     7. run() – Entry point for a thread
     8. sleep() – suspend thread for a specified time
     9. start() – start a thread by calling run() method

#### Some Important points to Remember

1. When we extend Thread class, we cannot override setName() and getName() functions, because they are declared final in Thread class.
2. While using sleep(), always handle the exception it throws.

*static* void **sleep**(long *milliseconds*) throws **InterruptedException**

* 1. Interthread Communication: -The **wait**(), **notify**(), **notifyaAll**() of Object class are implemented as final . All three methods can be called only from within **synchronized** context.
     1. wait() tells calling thread to give up monitor and go to sleep until some other thread enters the same monitor and call notify
     2. notify() wakes up a thread that called wait() on same object.
     3. notifyAll() wakes up all the thread that called wait() on same object.
  2. **Difference between wait() and sleep()**

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| called from synchronised block | no such requirement |
| monitor is released | monitor is not released |
| awake when notify() or notifyAll() method is called. | not awake when notify() or notifyAll() method is called |
| not a static method | static method |
| wait() is generaly used on condition | sleep() method is simply used to put your thread on sleep. |