Java Exception Handling Cheat Sheet

(javaconceptoftheday.com)

Basics

What is exception?

Exception is an abnormal condition which occurs during execution of a program and disrupts the normal flow of a program.

Ex : NumberFormatException, ArithmaticException, ArrayIndexOutOfBoundsException, ClassCastException, NullPointerException, StackOverflowError, OutOfMemoryError etc...

Exception Handling In Java :

Exceptions in Java are handled using try, catch and finally blocks.

```
try
{
    This block contains statements which may
    throw exceptions during run time.
}
catch(Exception e)
{
    This block handles the exceptions thrown by the
    try block.
}
finally
{
    This block is always executed whether an
    exception is thrown or not and thrown exception
    is caught or not.
}
```

Rules To Follow While Writing try-catch-finally Blocks :

- ✓ try, catch and finally blocks form one unit. There must be one try block and one or more catch blocks. finally block is optional.
- ✓ There should not be any statements in between the blocks.
- ✓ If there are multiple catch blocks, the order of catch blocks must be from most specific to general ones. i.e. lower classes in the hierarchy of exceptions must come first and higher classes later.

If try-catch-finally blocks are supposed to return a value :

- ✓ If finally block returns a value then try and catch blocks may or may not return a value.
- ✓ If finally block does not return a value then both try and catch blocks must return a value.
- ✓ finally block overrides return values from try and catch blocks.
- ✓ finally block will be always executed even though try and catch blocks are returning the control.

Frequently Occurring Exceptions

1) NullPointerException occurs when your application tries to access null object.

2) ArrayIndexOutOfBoundsException occurs when you try to access an array element with an invalid index i.e index greater than the array length or with a negative index.

3) NumberFormatException is thrown when you are trying to convert a string to numeric value like integer, float, double etc..., but input string is not a valid number.

4) ClassNotFoundException is thrown when an application tries to load a class at run time but the class with specified name is not found in the classpath.

5) ArithmeticException is thrown when an abnormal arithmetic condition arises in an application.

6) SQLException is thrown when an application encounters with an error while interacting with the database.

7) ClassCastException occurs when an object of one type can not be casted to another type.

8) IOException occurs when an IO operation fails in your application.

9) NoClassDefFoundError is thrown when Java Runtime System tries to load the definition of a class which is no longer available.

10) StackOverflowError is a run time error which occurs when stack overflows. This happens when you keep calling the methods recursively.

Types Of Exceptions

There are two types of exceptions in Java.

1. **Checked Exceptions** are the exceptions which are checked during compilation itself.
2. **Unchecked Exceptions** are the exceptions which are not checked during compilation. They occur only at run time.

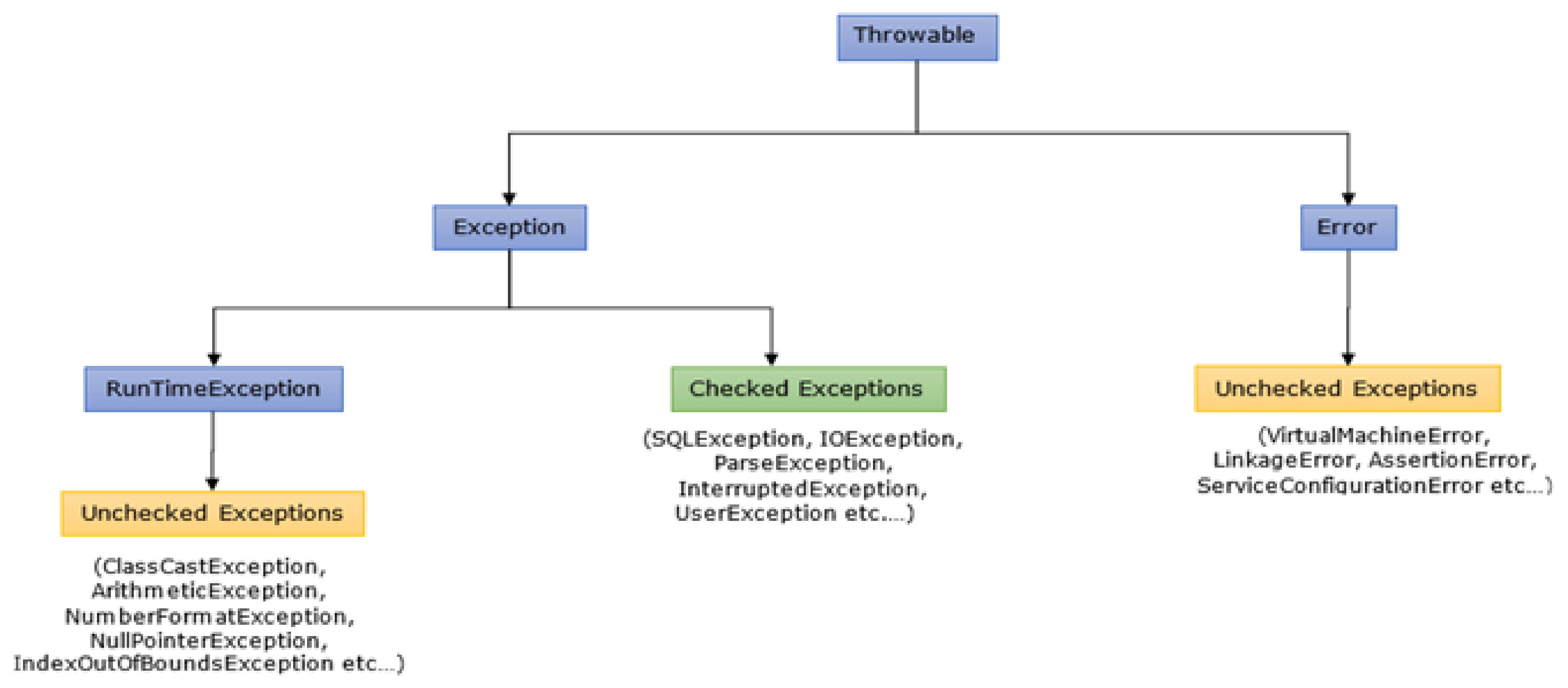
Checked Exceptions	**Unchecked Exceptions**
They are checked at compile time.	They are not checked at compile time.
They are compile time exceptions.	They are run time exceptions.
These exceptions must be handled properly either using try-catch blocks or using throws clause, otherwise compiler will throw error.	If these exceptions are not handled properly, compiler will not throw any error. But, you may get error at run time.
All the sub classes of java.lang.Exception (except sub classes of java.lang.RunTimeException) are checked exceptions.	All the sub classes of java.lang.RunTimeException and all the sub classes of java.lang.Error are unchecked exceptions.
Ex : FileNotFoundException, IOException, SQLException, ClassNotFoundException	Ex : NullPointerException, ArithmeticException, ClassCastException, ArrayIndexOutOfBoundsException

Hierarchy Of Exceptions

java.lang.Throwable is the super class for all type of errors and exceptions in Java.

It has two sub classes.

1. **java.lang.Error :** It is the super class for all types of errors in Java.
2. **java.lang.Exception :** It is the super class for all types of exceptions in Java.


Throwable
Exception
Error
RunTimeException
Checked Exceptions
Unchecked Exceptions
(SQLException, IOException, ParseException, InterruptedException, UserException etc....)
(VirtualMachineError, LinkageError, AssertionError, ServiceConfigurationError etc...)
Unchecked Exceptions
(ClassCastException, ArithmeticException, NumberFormatException, NullPointerException, IndexOutOfBoundsException etc...)

throw Keyword

throw keyword is used to throw an exception explicitly.

```
try
{
      throw InstanceOfThrowableType;
}
catch(InstanceOfThrowableType)
{

}
```

where, InstanceOfThrowableType must be an object of type Throwable or subclass of Throwable.

throws Keyword

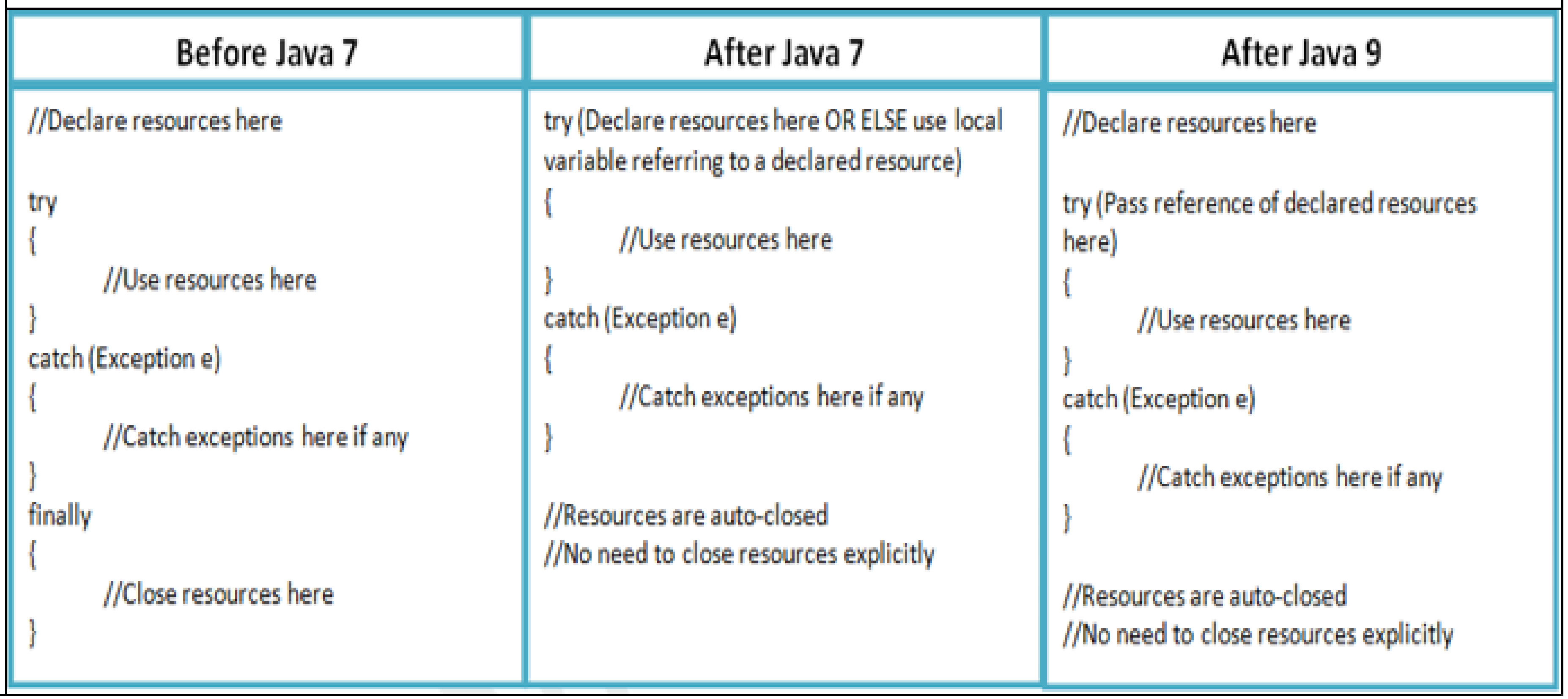
throws keyword is used to specify the exceptions that may be thrown by the method.

```
return_type method_name(parameter_list) throws
exception_list
{
    //some statements
}
```

where, exception_list is the list of exceptions that method may throw. Exceptions must be separated by commas.

Try-with Resources

Try with resources blocks are introduced from Java 7. In these blocks, resources used in try blocks are auto-closed. No need to close the resources explicitly. But, Java 7 try with resources has one drawback. It requires resources to be declared locally within try block. It doesn't recognize resources declared outside the try block. That issue has been resolved in Java 9.



Before Java 7	After Java 7	After Java 9

//Declare resources here
 	
try
{
//Use resources here
}
catch (Exception e)
{
//Catch exceptions here if any
}
finally
{
//Close resources here
}
	try (Declare resources here OR ELSE use local variable referring to a declared resource)
	{
	//Use resources here
	}
	catch (Exception e)
	{
	//Catch exceptions here if any
	}
	
	//Resources are auto-closed
	//No need to close resources explicitly
		//Declare resources here
		
		try (Pass reference of declared resources here)
		{
		//Use resources here
		}
		catch (Exception e)
		{
		//Catch exceptions here if any
		}
		
		//Resources are auto-closed
		//No need to close resources explicitly