edureka!

JAVA EXCEPTIONS CHEAT SHEET

Learn JAVA from experts at http://www.edureka.co

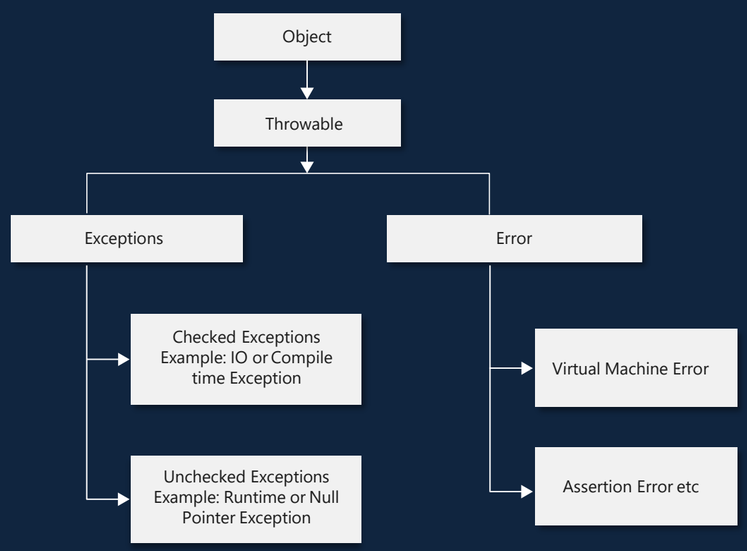
Exception Handling

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

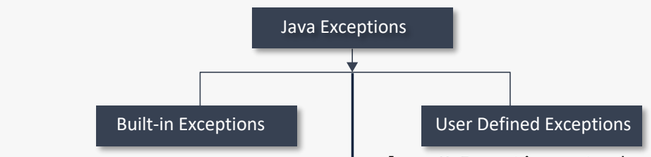
Error vs Exception

Error	Exception
1. Impossible to recover from error.	1. Possible to recover from Exception.
2. Errors are of type unchecked exception.	2. Exceptions can be checked type or unchecked type.
3. All errors are of type java.lang.error.	3. All exceptions are of type java.lang. Exception
4. They are not known to compiler. They happen at run time.	4. Checked exceptions are known to compiler where as unchecked exceptions are not known to compiler.
5. Errors are caused by the enviroment in which the application is running.	5. Exceptions are caused by the application.

Exception Hirerachy



Types of Exception in Java with Examples



1. Arithmetic Exception
2. ArrayIndexOutOfBoundException
3. ClassNotFoundException
4. FileNotFoundException
5. IOException
6. InterruptedException
7. NoSuchFieldException
8. NoSuchMethodException
9. NullPointerException
10. NumberFormatException
11. RuntimeException
12. StringIndexOutOfBoundsException

```
class MyException extends
  Exception{  String str1;
  MyException(String str2) {
       str1=str2;
  }
  public String toString(){
    return ("MyException Occurred:
 "+str1);
  }
  }
   class Example1{
    public static void main(String
     args[]){
     try{
      System.out.println("Start of try
      block");
      throw new MyException("Error
      Message");
       }
       catch(MyException exp){
        System.out.println("Catch
        Block");
        System.out.println(exp);
       }
    }
```

throw vs throws

Java throw example

```
void a(){
throw new ArithmeticException("Incorrect");}
```

Java throws example

```
void a()throws ArithmeticException {}
```

Java throw and throws example

```
void a()throws ArithmeticException{
throw new ArithmeticException("Incorrect");
}
```

Exception Handling Methods

try block

```
try{
//code that throws exception
}catch(Exception_class_Name){}
```

catch block

```
 public class Sampletrycatch1{
 public static void main(String args[])
 {
   int data=50/0;//throws exception
 System.out.println("remaning code");
 }
```

Multi catch block

```
public class SampleMultipleCatchBlock{
public static void main(String args[]){
try{
  int a[]=new int[5];  a[5]=30/0;
  }
  catch(ArithmeticException e)
      {System.out.println("task1  is completed");}
   catch(ArrayIndexOutOfBoundsException e)
      {System.out.println("task 2 completed");}
   catch(Exception e)
      {System.out.println("task 3 completed");}
System.out.println("remaining code");
}
}
```

Nested try block

```
 class Exception{
 public static void main(String args[]){
 try{
 try{
    System.out.println("going to divide");
    int b=59/0;
    }catch(ArithmeticException e){System.out.println(e);}
    try{
       int a[]=new int[5];  a[5]=4;
       }
 catch(ArrayIndexOutOfBoundsException e)
 {System.out.println(e);}
    System.out.println("other statement");
 }catch(Exception e)
 {System.out.println("Exception handeled");}
    System.out.println("casual flow");
 }
 }
```

Fundamentals of Java Exceptions

Basic Exception

```
public class ExceptionExample{
    public static void main(Stringargs[]){
        try{
             //code that raise exception
             int data=100/0;
          }catch(ArithmeticException e){System.out.println(e);}
            //rest of the program
            System.out.println("rest of the code...");
      }
}
```

Exception Methods

```
public String getMessage()
public Throwable getCause()
public String toString()
public void printStackTrace()
public StackTraceElement []
getStackTrace()
public Throwable fillInStackTrace()
```

Common Scenarios

1. ***ArithmeticException***

```
int a=50/0;
```

2. ***NullPointerException***

```
String a=null;
System.out.println(a.length());
```

3. ***NumberFormatException***

```
String s="abc";
int i=Integer.parseInt(s);
```

4. ***ArrayIndexOutOfBoundsException***

```
int a[]=new int[5];  a[10]=50;
```

Exception Methods

try	-	The "try" keyword is used to specify a block where we should place exception code.
catch	-	The "catch" block is used to handle the exception.
finally	-	The "finally" block is used to execute the important code of the program.
throw	-	The "throw" keyword is used to throw an exception.
throws	-	The "throws" keyword is used to declare exceptions.

Exception Handling with Method Overriding in Java

If the superclass method does not declare an exception

```
class Parent{
  void msg(){System.out.println("parent");}
  }
  class ExceptionChild extends Parent{
  void msg()throws IOException{
    System.out.println("ExceptionChild");
  }
  public static void main(String args[]){  Parent
   p=new ExceptionChild();
   p.msg();
  }
  }
```

Subclass overridden method declares parent exception

```
class Parent{
void msg()throwsArithmeticException
{System.out.println("parent");}
}
class ExceptionChild2 extends Parent{
void msg()throws Exception{
System.out.println("child");}
public static void main(String args[]){
Parent p=new ExceptionChild2();
try{
p.msg();
}catch(Exception e){}
}
```

final	finally	finalise
final is a keyword Used to apply restrictions on class, methods and variables • Final class cannot be inherited • Final method cant be overridden' • Final variable cant be changed	finally is a block Used to place important code It will be executed whether the exception is handled or not	finalise is a method Used to perform clean up processing just before the object is garbage collected

finally block

```
class SampleFinallyBlock{
 public static void main(String args[]){
 try{
  int data=55/5;
  System.out.println(data);
  }
 catch(NullPointerException e){System.out.println(e);}
 finally{System.out.println("finally block is executed");}
 System.out.println("remaining code");
 }
}
```