Built-in Exceptions

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

S. No.	Exception	Description
1.	**ArithmeticException**	Thrown when a problem in arithmetic operation is noticed by the JVM.
2.	**ArrayIndexOutOfBoundsException**	Thrown when you access an array with an illegal index.
3.	**ClassNotFoundException**	Thrown when you try to access a class which is not defined
4.	**FileNotFoundException**	Thrown when you try to access a non-existing file.
5.	**IOException**	Thrown when the input-output operation has failed or interrupted.
6.	**InterruptedException**	Thrown when a thread is interrupted when it is processing, waiting or sleeping
7.	**IllegalAccessException**	Thrown when access to a class is denied
8.	**NoSuchFieldException**	Thrown when you try to access any field or variable in a class that does not exist
9.	**NoSuchMethodException**	Thrown when you try to access a non-existing method.
10.	**NullPointerException**	Thrown when you refer the members of a null object
11.	**NumberFormatException**	Thrown when a method is unable to convert a string into a numeric format
12.	**StringIndexOutOfBoundsException**	Thrown when you access a String array with an illegal index.

A. Checked Exceptions:

- ✓ Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.
- ✓ Checked Exceptions forces programmers to deal with the exception that may be thrown.
- ✓ The compiler ensures whether the programmer handles the exception using try.. catch () block or not. The programmer should have to handle the exception; otherwise, compilation will fail and error will be thrown.

Example:

1. ClassNotFoundException
2. CloneNotSupportedException
3. IllegalAccessException,
4. MalformedURLException.
5. NoSuchFileException
6. NoSuchMethodException
7. IOException

Example Program: (Checked Exception)

`FileNotFoundException` is a checked exception in Java. Anytime, we want to read a file from filesystem, Java forces us to handle error situation where file may not be present in place.

Without try-catch

```
import java.io.*;

public class CheckedExceptionExample {
public static void main(String[] args)
{
    FileReader file = new FileReader("src/somefile.txt");
}
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Unhandled exception type FileNotFoundException
```

To make program able to compile, you must handle this error situation in try-catch block.

Below given code will compile absolutely fine.

With try-catch

```
import java.io.*;
public class CheckedExceptionExample {
public static void main(String[] args) {
try {
@SuppressWarnings("resource")
FileReader file = new FileReader("src/somefile.java");
System.out.println(file.toString());
}
catch(FileNotFoundException e){
System.out.println("Sorry...Requested resource not availabe...");
} }
}
```

Output:

```
Sorry...Requested resource not availabe...
```

B. Unchecked Exceptions(RunTimeException):

- ✓ The **unchecked** exceptions are just opposite to the **checked** exceptions.
- ✓ Unchecked exceptions are not checked at compile-time rather they are checked at runtime.
- ✓ The compiler doesn't force the programmers to either catch the exception or declare it in a throws clause.
- ✓ In fact, the programmers may not even know that the exception could be thrown.

Example:

1. ArrayIndexOutOfBoundsException
2. ArithmeticException
3. NullPointerException.

Example: Unchecked Exception

Consider the following Java program. It compiles fine, but it throws *ArithmeticException* when run. The compiler allows it to compile, because *ArithmeticException* is an unchecked exception.

```
class Main {
    public static void main(String args[]) {
        int x = 0;
        int y = 10;
        int z = y/x;
    }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Main.main(Main.java:5)
```

Example 1: NullPointer Exception

```
//Java program to demonstrate NullPointerException
class NullPointer_Demo
{
  public static void main(String args[])
  {
    try {
       String a = null; //null value
       System.out.println(a.charAt(0));
    } catch(NullPointerException e) {
       System.out.println("NullPointerException..");
```

```
    }
  }
}
```

Output:

NullPointerException..

Example 2: NumberFormat Exception

```
// Java program to demonstrate NumberFormatException
class NumberFormat_Demo
{
   public static void main(String args[])
   {
     try {
       // "akki" is not a number
       int num = Integer.parseInt ("akki") ;
       System.out.println(num);
     } catch(NumberFormatException e) {
         System.out.println("Number format exception");
       }
    }
}
```

Output:

Number format exception

3.5.2: USER-DEFINED EXCEPTIONS (CUSTOM EXCEPTIONS)

Exception types created by the user to describe the exceptions related to their applications are known as **User-defined Exceptions** or **Custom Exceptions.**

To create User-defined Exceptions:

1. Pick a self-describing ***Exception** class name.
2. Decide if the exception should be checked or unchecked.
   - ✓ Checked : **extends Exception**
   - ✓ Unchecked: **extends RuntimeException**
3. Define constructor(s) that call into super class constructor(s), taking message that can be displayed when the exception is raised.
4. Write the code that might generate the defined exception inside the try-catch block.
5. If the exception of user-defined type is generated, handle it using **throw clause** as follows:

   throw ExceptionClassObject;

Example:

The following program illustrates how user-defined exceptions can be created and thrown.

```
public class EvenNoException extends Exception
{
   EvenNoException(String str)
   {
      super(str);  // used to refer the superclass constructor
   }

   public static void main(String[] args)
   {
         int arr[]={2,3,4,5};
         int rem;
         int i;
         for(i=0;i<arr.length;i++)
         {
          rem=arr[i]%2;
          try
          {
            if(rem==0)
            {
               System.out.println(arr[i]+" is an Even Number");
            }
            else
            {
               EvenNoException exp=new EvenNoException(arr[i]+" is
               not an Even Number");
               throw exp;
            }
          }
          catch(EvenNoException exp)
           {
            System.out.println("Exception thrown is "+exp);
           }
      } // for loop
   } // main()
} // class
```

Output:

0 is an Even Number
Exception thrown is EvenNoException: 3 is not an Even Number
4 is an Even Number

Exception thrown is EvenNoException: 5 is not an Even Number

Program Explanation:

In the above program, the **EvenNumberException** class is created which inherits the **Exception** super class. Then the constructor is defined with the call to the super class constructor. Next, an array **arr** is created with four integer values. In the **main(),** the array elements are checked one by one for even number. If the number is odd, then the object of EvenNumberException class is created and thrown using **throw** clause. The EvenNumberException is handled in the catch block.

Comparison Chart - final Vs. finally Vs. finalize

Basis for comparison	final	finally	finalize
Basic	Final is a "Keyword" and "access modifier" in Java.	Finally is a "block" in Java.	Finalize is a "method" in Java.
Applicable	Final is a keyword applicable to classes, variables and methods.	Finally is a block that is always associated with try and catch block.	finalize() is a method applicable to objects.
Working	(1) Final variable becomes constant, and it can't be reassigned. (2) A final method can't be overridden by the child class. (3) Final Class can not be extended.	A "finally" block, clean up the resources used in "try" block.	Finalize method performs cleans up activities related to the object before its destruction.
Execution	Final method is executed upon its call.	"Finally" block executes just after the execution of"try-catch" block.	finalize() method executes just before the destruction of the object.

Example	class FinalExample{
	public static void main(String[] args){
	final int x=100;
	x=200;//Compile Time Error
	}}
		class FinallyExample{
		public static void main(String[] args){
		try{
		int x=300;
		}catch(Exception e){System.out.println(e);}
		finally{
		System.out.println("finally block is executed");
		}
		}}
			class FinalizeExample{
			public void finalize(){System.out.println("finalize called");}
			public static void main(String[] args){
			FinalizeExample f1=new FinalizeExample();
			FinalizeExample f2=new FinalizeExample();
			f1=null;
			f2=null;
			System.gc();
			}}