# Overview of Prdefined Exceptions:

# ArithmeticException

In java applications, when we have a situation like a number is devided by zero then JVM will rise Arithmetic Exception.

**EX:**

class Test

{

public static void main(String[] args)

{

int i=10;

int j=0;

float f=i/j;

System.out.println(f);

}

}

If we run the above program then JVM will provide ArithmeticException with the following Exception message:

**Exception in thread "main" java.lang.ArithmeticException: / by zero at Text.main(Test.java:7)**

**The above exception message is divided into the following three parts:**

1. **Exception Name: java.lang.ArithmeticException**
2. **Exception Descrioptioun: / by zero**
3. **Exception Location : Test.java:7**

# NullPointerException:

In java applications, when we access any instance variable and instance methods by using a reference variable contains null value then JVM will rise NullPointerException.

**EX:**

class Test

{

public static void main(String[] args)

{

java.util.Date d=null;

System.out.println(d.toString());

}

}

**If we run the above code then JVM will provide the following exception details:**

**1.Exception Name : java.lang.NullPointerException**

**2.Exception Description: --- No description----**

**3.Exception Location: Test.java:6**

# ArrayIndexOutOfBoundsException:

In java applications, when we insert an element to an array at a particular index value and when we are trying to access an element from an array at a particular index value and if the specified index value is in out side range of the arrays size then JVM will rise an exception like "ArrayIndexOutOfBoundsException".

**EX:**

class Test

{

public static void main(String[] args)

{

int [] a={1,2,3,4,5};

System.out.println(a[10]);

}

}

If we run the above program then JVM will rise an exception with the following details.

**Exception Name : java.lang.ArrayindexOutOfBoundsException Exception Description: 10**

**Exception Location: Test.java: 6**

# 4.ClassCastException:

In java applications, we are able to keep sub class object reference value in super class reference variable, but, we are unable to keep super class object reference value in sub class reference variable. If we are trying to keep super class object reference value in sub class reference variable then JVM will rise an exception like "java.lang.ClassCastException".

**EX:**

class A

{

}

class B extends A

{

}

class Test

{

public static void main(String[] args)

{

A a=new A();

B b=(B)a;

}

}

**If we run the above code then JVM will rise an exception with the following details.**

Exception Name: java.lang.ClassCastException Exception Description: A can not be cast to B Excepttion location: Test.java: 12

# 5.ClassNotFoundException:

In java applications, if we want to load a particular class bytecode to the memory with out creating object then we will use the following method from java.lang.Class class.

public static Class forName(String class\_Name)throws ClassNotFoundException

**EX: Class c=Class.forName("A");**

**When JVM encounters the above instuction, JVM will search for A.class file at current location, at java predefined library and at the locations refered by "classpath" environment variable, if the required A.class file is not identified at all the locations then JVM will rise ClassNotFoundException.**

**EX:**

class A

{

static

{

System.out.println("Class Loading");

}

}

class Test

{

public static void main(String[] args) throws Exception

{

Class c=Class.forName("AAA");

}

}

**If we run this code the JVM will provide the following exception details.**

**Exception Name: java.lang.ClassNotFoundException**

**Exception Description: AAA**

**Exception Location: Test.java: 12**

(6,7) InstantiationException and IllegalAccessException:

In java applications, if we load class bytecode by using "Class.forName(-)" method then to create object explicitly we have to use the following method from java.lang.Class class.

**public Object newInstance()**

**EX: Object obj=c.newInstance()**

**When JVM encouter the above code then JVM will search for 0-arg constructor and non-private constructor in the loaded class. if 0-arg constructor is not availabnle, if parameterized construcvtor is existed then JVM will rise "java.lang.instantiationException".If non-private constructor is not available, if private constructor is available then JVM will rise "java.lang.IllegalAccessException".**

**EX1**:

class A

{

static

{

System.out.println("Class Loading");

}

A(int i)

{

System.out.println("Object Creating");

}

}

class Test

{

public static void main(String[] args) throws Exception {

Class c=Class.forName("A");

Object obj=c.newInstance();

}

}

**If run the above code then JVM will provide an exception with the following details.**

**Exception Name : java.lang.InstantiationException**

**Exception Description: A**

**Exception Location:Test.java: 17**

**EX:**

class A

{

static

{

System.out.println("Class Loading");

}

private A()

{

System.out.println("Object Creating");

}

}

class Test

{

public static void main(String[] args) throws Exception {

Class c=Class.forName("A");

Object obj=c.newInstance();

}

}

**If we run the above code then JVM will rise an exception with the following details**

**Exception Name : java.lang.IllegalAccessException**

**Exception Decrition: Test class can not access the members of class A with the modifier “private”**

**Exception Location" Test.java: 17**