Day_11_OOPJ_Sanket_Shalukar

Tuesday, September 09, 2025 10:24 AM

Topics are in the Day_11

1. Exception Handling
2. Unchecked Exception
3. Checked Exception
4. Wrapper Class
5. Collections

Java Exception Handling

Exception handling in Java is an effective mechanism for managing runtime errors to ensure the application's regular flow is maintained. Some Common examples of exceptions include ClassNotFoundException, IOException, SQLException, RemoteException, etc. By handling these exceptions, Java enables developers to create robust and fault-tolerant applications.

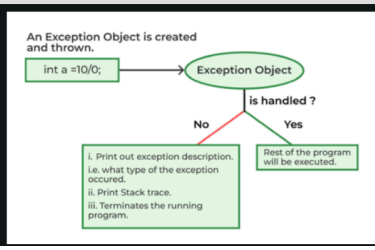
Example: Showing an arithmetic exception or we can say a divide by zero exception.

```
import java.io.*;

class Geeks {
    public static void main(String[] args)
    {
        int n = 10;
        int m = 0;

        int ans = n / m;

        System.out.println("Answer: " + ans);
    }
}
```

Output:

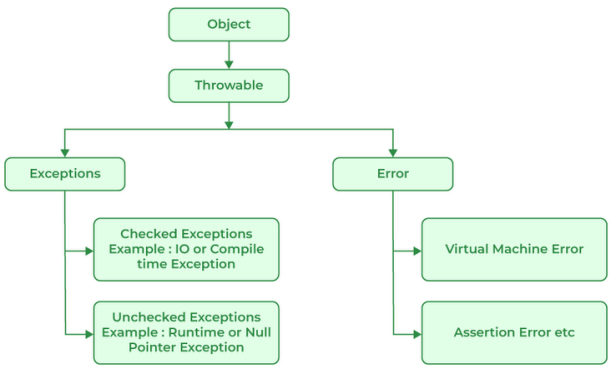
```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at Main.main(Main.java:11)

Process finished with exit code 1
```

```
Error: Division by zero is not allowed!
Program continues after handling the exception.
```



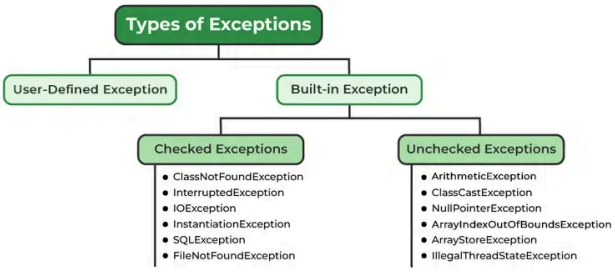
Java Exception Hierarchy :

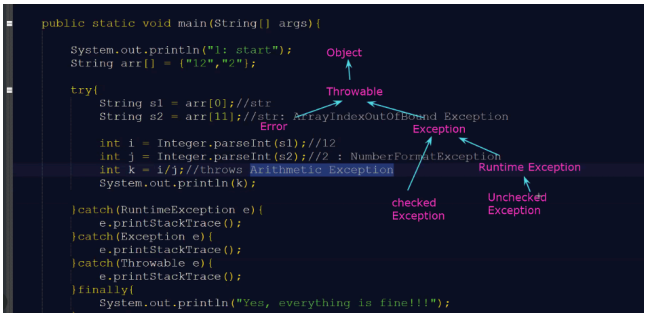
1. Exception.
2. Error

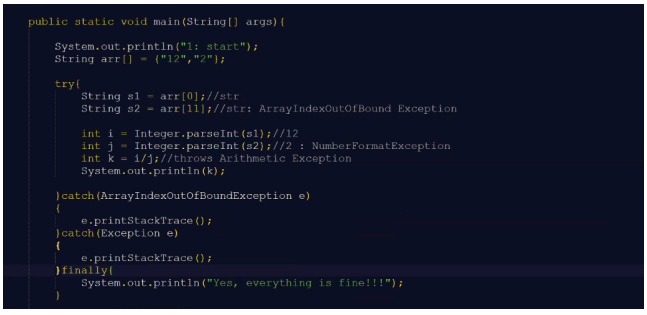


Major Reasons Why an Exception Occurs, Exceptions can occur due to several reasons, such as:

1. Invalid user input
2. Device failure
3. Loss of network connection
4. Physical limitations (out-of-disk memory)
5. Code errors
6. Out of bound
7. Null reference
8. Type mismatch
9. Opening an unavailable file
10. Database errors
11. Arithmetic errors

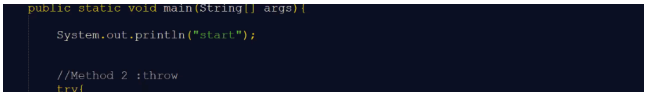






Abrupt Termination :

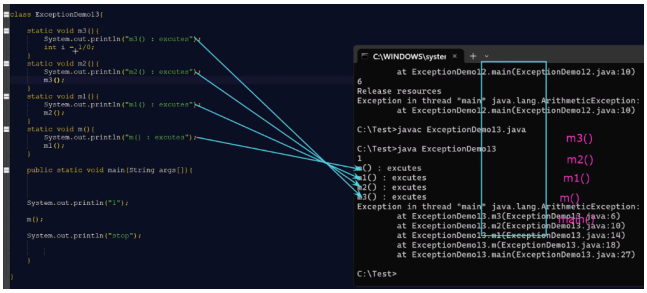
When the program **stops suddenly** without completing all remaining statements, usually because of an **unhandled exception**.



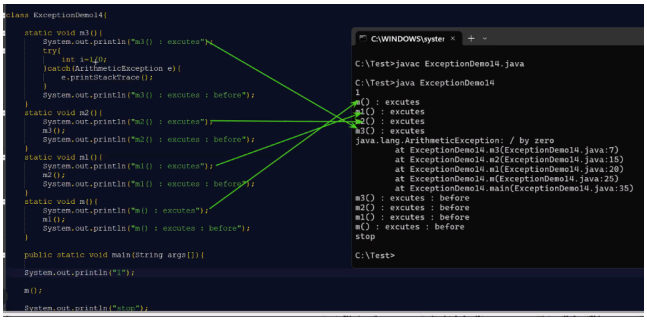
```
        System.out.println("aaaaaaa");

        throw new ArithmeticException();//i/0 (background)
        //throw new NullPointerException();///d1 = null;(background)

        //System.out.println("bbbbbb");

    }catch(NullPointerException e){
        e.printStackTrace();
    }catch(ArithmeticException e){
        e.printStackTrace();
    }
    finally{
        System.out.println("Bhai Resources ko release kar do!!!");
    }
    System.out.println("stop");

}
```

Arithmatic Exception

```
    System.out.println("start");


    //Method 2 :throw
    try{
        System.out.println("aaaaaaa");

        //int i =i/0; : Method1
        throw new ArithmeticException();//Method 2

        //ExceptionDemo10 d1 = null;
        throw new NullPointerException();//d1 = null;(background)

        //System.out.println("bbbbbb");

    }catch(NullPointerException e){
        e.printStackTrace();
    }catch(ArithmeticException e){
        e.printStackTrace();
    }
    finally{
        System.out.println("Bhai Resources ko release kar do!!!");
    }
    System.out.println("stop");
```




```
class ExceptionDemo14{

    static void m3(){
        System.out.println("m3() : excutes");
        try{
        System.out.println("m3() : excutes : before");
    }
    static void m2(){
        System.out.println("m2() : excutes");
        m3();
        System.out.println("m2() : excutes : before");
    }
    static void m1(){
        System.out.println("m1() : excutes");
        m2();
        System.out.println("m1() : excutes : before");
    }
    static void m(){
        System.out.println("m() : excutes");
        m1();
        System.out.println("m() : excutes : before");
    }

    public static void main(String args[]){
```



```
class ExceptionDemo16{


    public static void main(String args[]){

        throw new NullPointerException();

    System.out.println("1");

    intarr[] = {1,2,3,4,5};
    arr[10] =10;//ArrayIndexOutOfBounds Exception

    String str = "abcd";
    char c = str.charAt(6);//StringIndexOutOfBounds Exception


    System.out.println("stop");

    }
```

```
}
```

```
import java.util.*;
import java.io.*;
class ChExceptionDemo1{


    public static void main(String args[]){
    System.out.println("1");

        try{
            System.exit(0);
        throw new NullPointerException();//int i =1/0;

        }catch(Exception e){
            //System.exit(0);
            e.printStackTrace();

        }finally{
            System.out.println("Finally .....");
        }


    System.out.println("stop");
```

```
import java.util.*;
import java.io.*;

class ChExceptionDemo1{


    public static void main(String args[]){
        System.out.println("start");
        //int i = Integer.parseInt("ab");//
        try{
            int j =System.in.read();//Chec
        }catch(IOException e){
            e.printStackTrace();
        }


    System.out.println("stop");

    }
}
```

```
C:\WINDOWS\syster
                int j =System.in.read();
                        ^
1 error

C:\Test>javac ChExceptionDemo1.java

C:\Test>java ChExceptionDemo1
start
a
stop

C:\Test>
```

```
public static void main(String args[]){
    System.out.println("1");
    m1();

}

static void m1(){
    System.out.println("2");
    System.out.println("m1() executed");
    m2();
    System.out.println("3");
}

static void m2(){
    System.out.println("4");
    System.out.println("m2() executed");
    System.out.println("Enter any character:");
    try{
        System.out.println("5");
        int j = System.in.read();
        System.out.println("Charcter="+(char)j);
        System.out.println("6");
    }catch(IOException e){
        e.printStackTrace();
        System.out.println("7");
    }
    System.out.println("8");
```

```
C:\WINDOWS\syster
m2() executed
Enter any character:
x
Charcter=x

C:\Test>javac ChExceptionDemo4.java

C:\Test>java ChExceptionDemo4
1
2
m1() executed
4
m2() executed
Enter any character:
5
Charcter=c
6
8
3

C:\Test>
```

I/O Exception

- **Type**: Checked Exception
- **Package**: `java.io`
- **Meaning**: Represents **any Input/Output failure**.
- It's a **general class** for all I/O errors.
- Examples:
- File not found
- Read/write failure
- Network issues while reading from a stream
- Disk not accessible

File NotFound Exception

- **Type**: Checked Exception
- **Package**: `java.io`
- **Parent**: Subclass of `IOException`
- **Meaning**: More **specific error**, when a file you try to open does **not exist** or is **inaccessible**.
- Example:
- `new FileReader("abc.txt");` when `abc.txt` does not exist

Wrapper Classes:

- In Java, every **primitive data type** (like `int`, `char`, `double`) has a corresponding **wrapper class** (`Integer`, `Character`, `Double`, etc.).

- Wrapper classes allow primitives to be used as **objects**, which is useful in collections (like `ArrayList`) and for using Java's built-in methods.

Uses

- Every primitive type in Java has a corresponding wrapper class:

- `int` → `Integer`
- `char` → `Character`
- `double` → `Double`
- Wrapper classes allow primitives to be used as objects, which is useful in collections like ArrayList.

Autoboxing:

- Automatic conversion of a primitive into its corresponding wrapper class object.
- Example: If you have an `int` value 10 and assign it to an `Integer` variable, Java automatically converts it into an `Integer` object.
- **Autoboxing** is the automatic conversion of a **primitive** into its corresponding **wrapper class object**.
- Example: Converting `int` to `Integer` automatically when needed.

Unboxing:

- Automatic conversion of a wrapper class object back to its primitive type.
- Example: If you have an `Integer` object with value 20 and assign it to an `int` variable, Java automatically converts it into a primitive `int`.
- **Unboxing** is the reverse process: converting a **wrapper object** back into its **primitive type**.

Collections:

- A **Collection** is a **framework in Java** used to **store, manage, and manipulate groups of objects**.
- It provides **ready-to-use data structures** like lists, sets, and queues, which makes programming easier.
- Collections can grow or shrink dynamically, unlike arrays which have fixed size.

Main Interfaces in Collections Framework:

1. **List** – An ordered collection that allows duplicates.
- Examples: ArrayList, LinkedList, Vector
2. **Set** – A collection that **does not allow duplicates**.
- Examples: HashSet, LinkedHashSet, TreeSet
3. **Queue** – A collection used to hold elements before processing, usually in **FIFO** order.
- Examples: PriorityQueue, LinkedList
4. **Map** – Stores data as **key-value pairs** (not part of Collection interface but part of Collections Framework).
- Examples: HashMap, LinkedHashMap, TreeMap