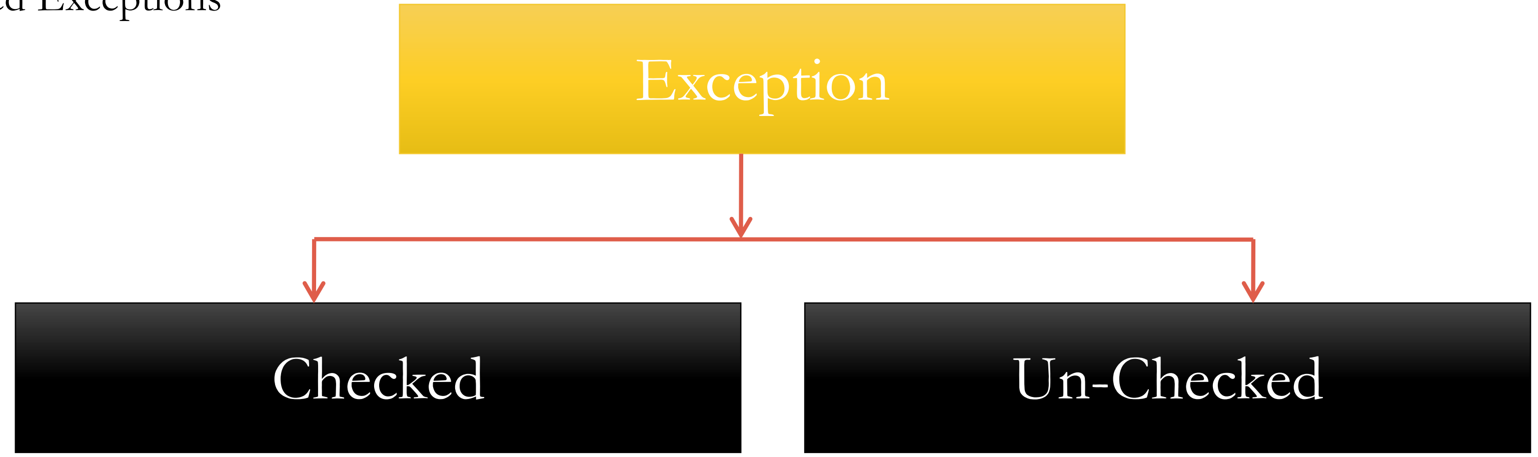
Java

Agenda

- Exception Handling

Java Exceptions

- Exception is an abnormal condition.
- In java, exception is an event that disrupts the normal flow of the program.
- There are two types of exceptions.
  1. Checked Exceptions
  2. Un-checked Exceptions



Un Checked Exceptions

- Exceptions that are NOT checked by compiler are called Un-Checked Exceptions.
- Un checked Exceptions successfully compiled by Java compiler.
- At run time it throws exception.
- Examples:
  - ArithmeticException
  - NullPointerException
  - NumberFormatException
  - ArrayIndexOutOfBoundsException

Common Un-Checked exceptions

`int a=50/0`	ArithmeticException

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

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

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

Checked Exceptions

- Exceptions that are checked by compiler are called Checked Exceptions.
- If a program contains checked-Exception code is not compiled.
- Examples:
  - InterruptedException
  - IOException
  - FileNotFoundException etc.

Common Checked exceptions

```
Thread.sleep(3000);
```

InterruptedException

```
FileReader fr = new FileReader("C:\\Test.txt");
BufferedReader bfr = new BufferedReader(fr);
System.out.println(bfr.readLine());
```

FileNotException

IOException

Java Exception Handling Keywords

- try
- catch
- finally
- throws

Java try..catch block

- Java try block is used to enclose the code that might throw an exception.
- It must be applied **at statement level within the method**.
- Java try block must be followed by either catch or finally block.
- Used for both **Un-checked and Checked Exceptions**.
- Java catch block is used to handle the Exception. It must be used after the try block only.
- You can use multiple catch block with a single try.

```
try{
//code that may throw exception
}
catch(Exception_class_Name ref)
{
//recovery code
}
```

Problem without exception handling

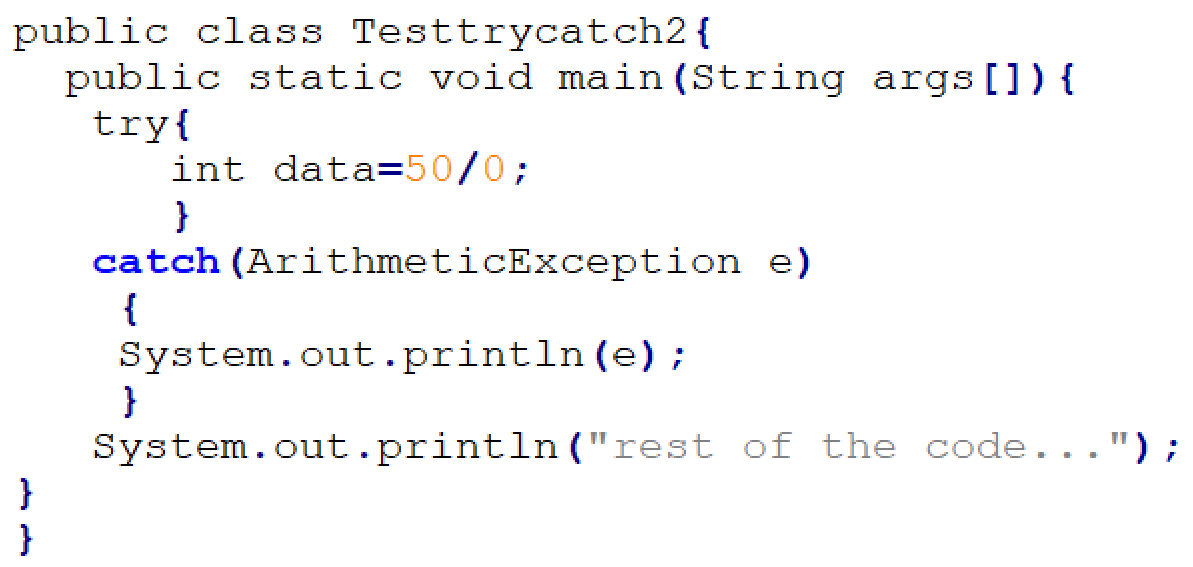
- Output: Exception in thread main java.lang.ArithmeticException:/ by zero

```
public class Testtrycatch1{
  public static void main(String args[])
    {
      int data=50/0;  //may throw exception
      System.out.println("rest of the code...");
    }
}
```

Solution by exception handling

Output: Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...



```
public class Testtrycatch2{
  public static void main(String args[]){
   try{
      int data=50/0;
      }
   catch(ArithmeticException e)
    {
    System.out.println(e);
    }
   System.out.println("rest of the code...");
}
}
```

Java Multi catch block

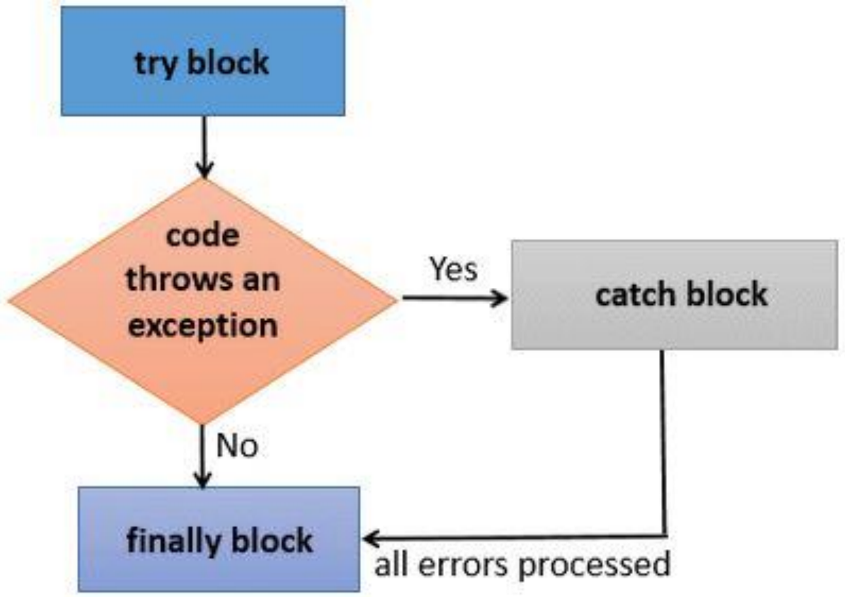
- If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

```
public class TestMultipleCatchBlock{
  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("common task completed");}

   System.out.println("rest of the code...");
 }
}
```

Java finally block

- **Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc.
- Java finally block is always executed whether exception is handled or not.
- Java finally block follows try or catch block.



Usage of Java finally

- Cases
  1. Exception doesn't occur.
  2. Exception occurs and not handled.
  3. Exception occurs and handled.

Case 1: Java finally example where exception doesn't occur

```
class TestFinallyBlock{
  public static void main(String args[])
  {
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e)
  {
   System.out.println(e);
  }
  finally
  {
  System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Case 2: Java finally example where exception occurs and not handled.

- Output:finally block is always executed
- Exception in thread main java.lang.ArithmeticException:/ by zero

```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(NullPointerException e)
  {
  System.out.println(e);
  }
  finally
  {
  System.out.println("finally block is always executed");
  }
  System.out.println("rest of the code...");
  }
}
```

Case 3: Java finally example where exception occurs and handled.

Output: Exception in thread main java.lang.ArithmeticException:/ by zero finally block is always executed rest of the code...

```
public class TestFinallyBlock2{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){
  System.out.println(e);
  }
  finally
  {
  System.out.println("finally block is always executed");
  }
  System.out.println("rest of the code...");
  }
}
```

throws

- Used for only Checked Exceptions.
- It should be applied at Method level.

throws – Example1

```
public class Test {
public static void main(String[] args) throws InterruptedException
{
System.out.println("Test started");
System.out.println("Test is in progress");
Thread.sleep(3000); // InterruptedException
System.out.println("Test is completed");
System.out.println("Test is exited");
}
}
```

throws – Example2

```
public class Test {

public static void main(String[] args) throws IOException
{
FileReader fr = new FileReader("C:\\Test.txt"); //FileNotException
BufferedReader bfr = new BufferedReader(fr);
System.out.println(bfr.readLine());    //IOException
}
}
```

	Un-Checked	Checked	Method Level	Within the method
Try..Catch	Y	Y	N	Y
throws	N	Y	Y	N

Assingment

1. Write a java program for the following and handle exceptions by using try..catch and finally blocks.
   - Any number divide by zero.
   - int a[]=null;
   - a.length
   - String s="abc";
   - int i=Integer.parseInt(s);

2. Write a java program to handle IO Exception by using *throws*.