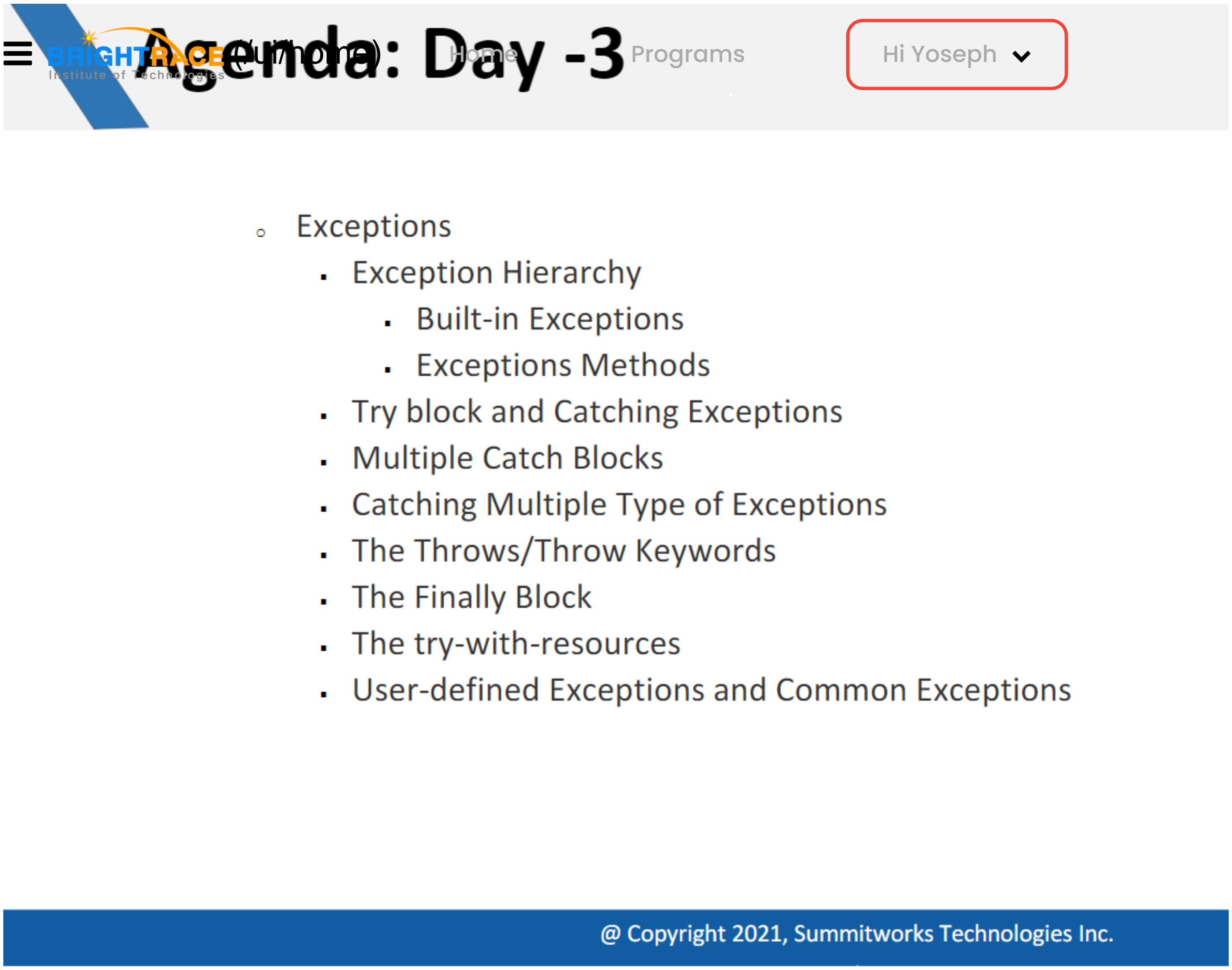
Toggle toolbar BRIGHTRACE Institute of Technologies (/ui/home) Home Programs Hi Yoseph

Java Spring & AWS
Sep 7, 2021 - Oct 8, 2021
Monday to Friday
9:30 AM ET - 4:30 PM ET

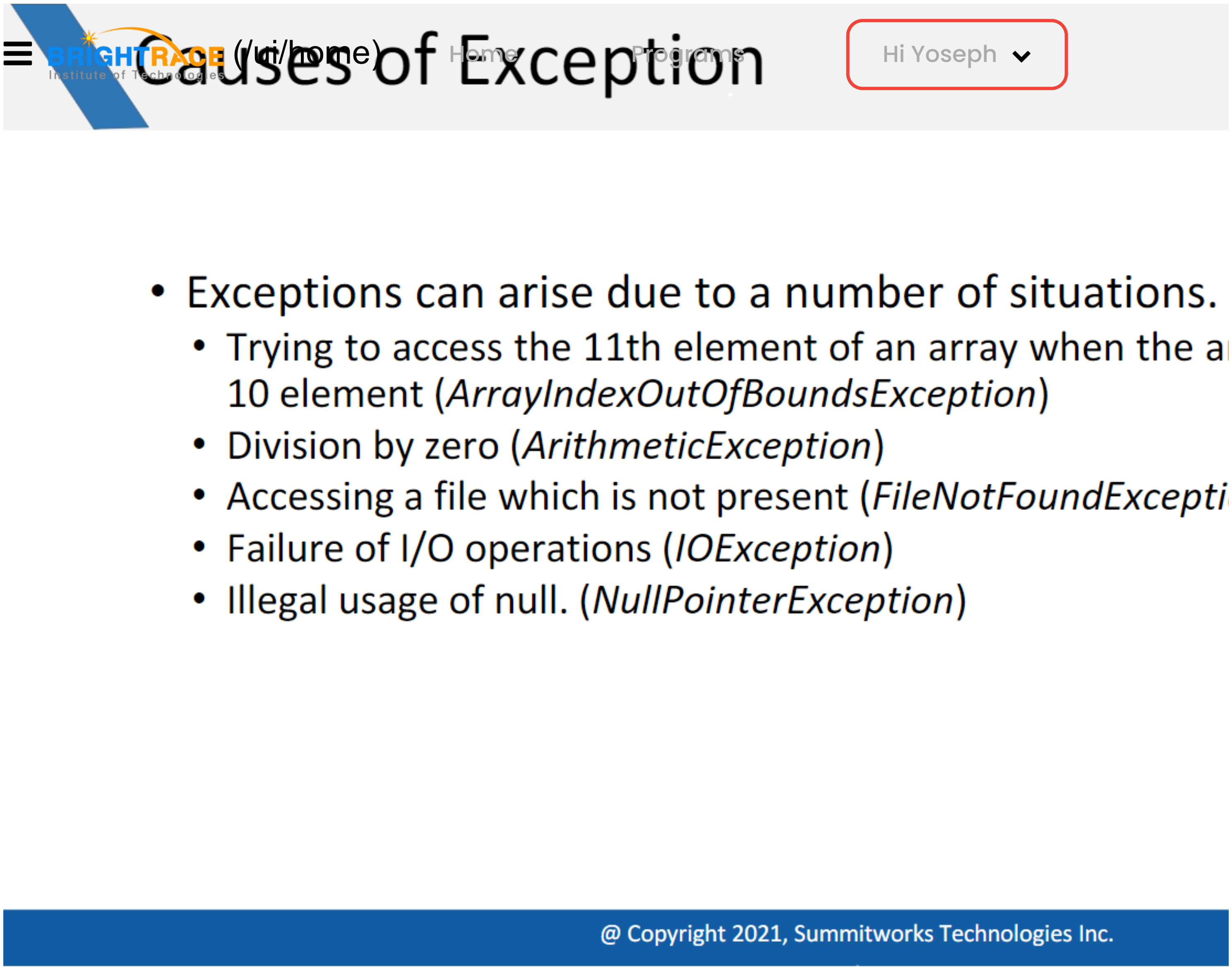


Agenda: Day -3

- Exceptions
  - Exception Hierarchy
    - Built-in Exceptions
    - Exceptions Methods
  - Try block and Catching Exceptions
  - Multiple Catch Blocks
  - Catching Multiple Type of Exceptions
  - The Throws/Throw Keywords
  - The Finally Block
  - The try-with-resources
  - User-defined Exceptions and Common Exceptions

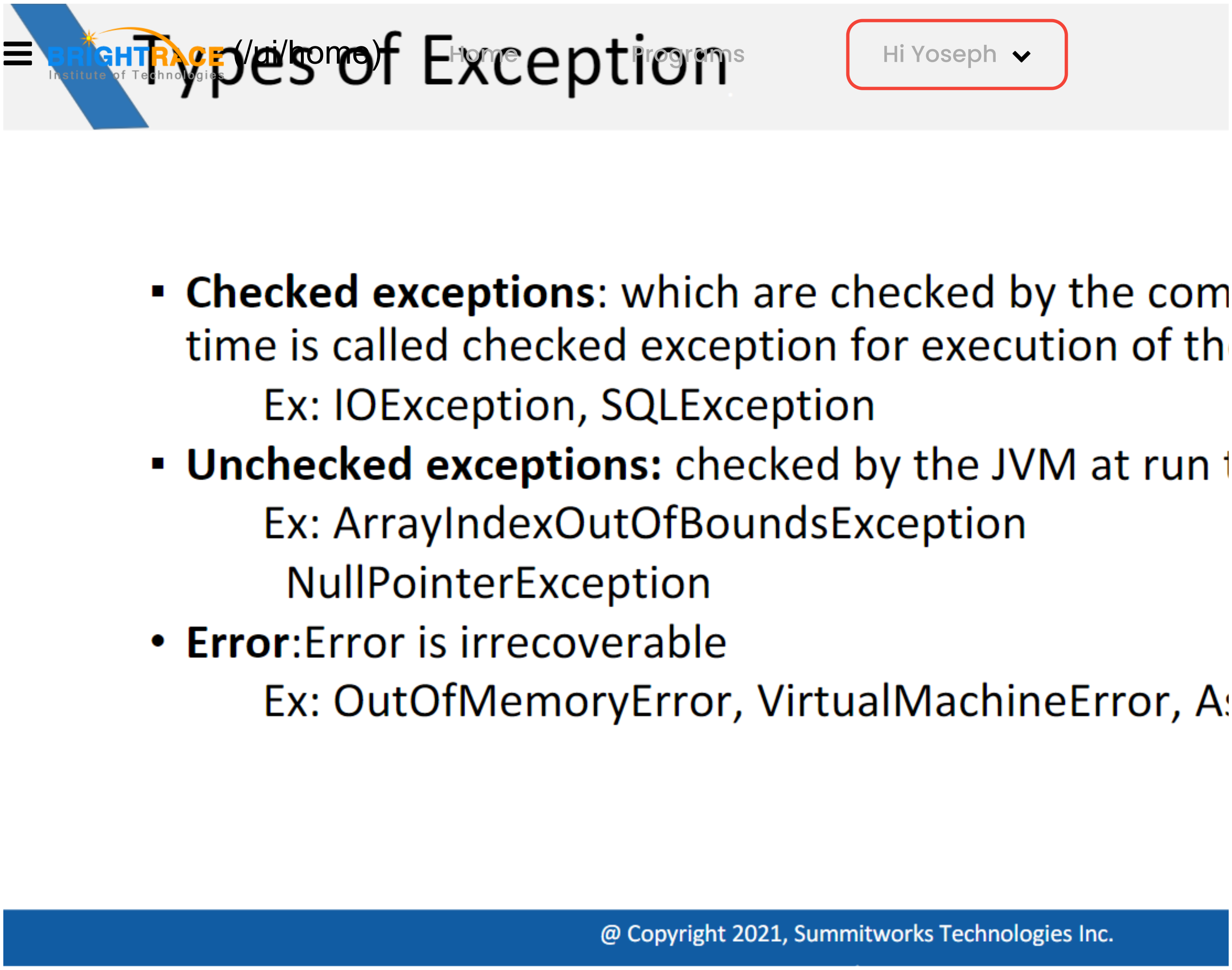
Exception

- are usually used to denote something unusua conform to the standard rules.
- In programming, exceptions are events that a occurrence of unexpected behaviour in cert disrupting the normal execution of a program.



Causes of Exception

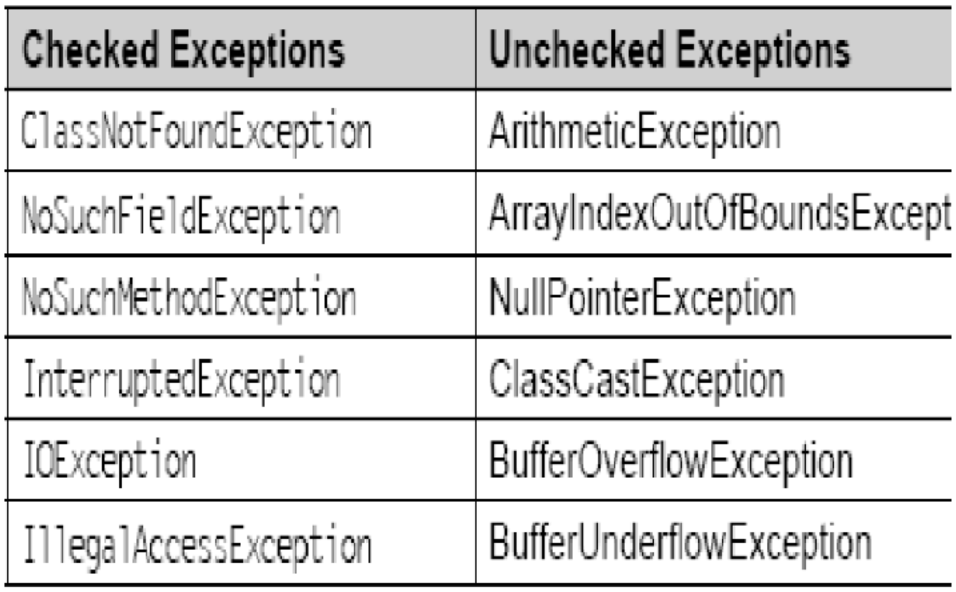
- Exceptions can arise due to a number of situations.
  - Trying to access the 11th element of an array when the a 10 element (*ArrayIndexOutOfBoundsException*)
  - Division by zero (*ArithmeticException*)
  - Accessing a file which is not present (*FileNotFoundExcepti*
  - Failure of I/O operations (*IOException*)
  - Illegal usage of null. (*NullPointerException*)



Types of Exception

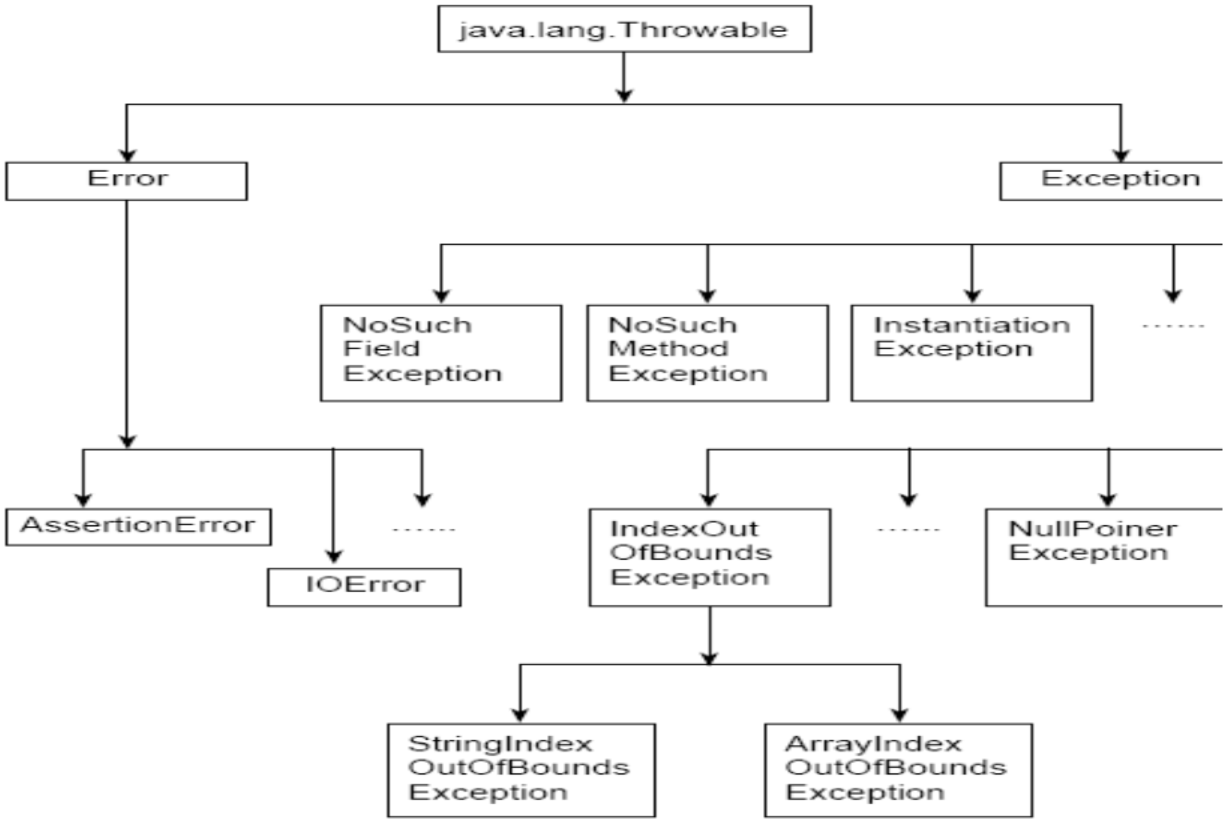
- **Checked exceptions**: which are checked by the com time is called checked exception for execution of th
  - Ex: IOException, SQLException
- **Unchecked exceptions:** checked by the JVM at run t
  - Ex: ArrayIndexOutOfBoundsException
  - NullPointerException
- **Error**:Error is irrecoverable
  - Ex: OutOfMemoryError, VirtualMachineError, As

Types of Exceptions



Checked Exceptions	Unchecked Exceptions
ClassNotFoundException	ArithmeticException
NoSuchFieldException	ArrayIndexOutOfBoundsExcept
NoSuchMethodException	NullPointerException
InterruptedException	ClassCastException
IOException	BufferOverflowException
IllegalAccessException	BufferUnderflowException

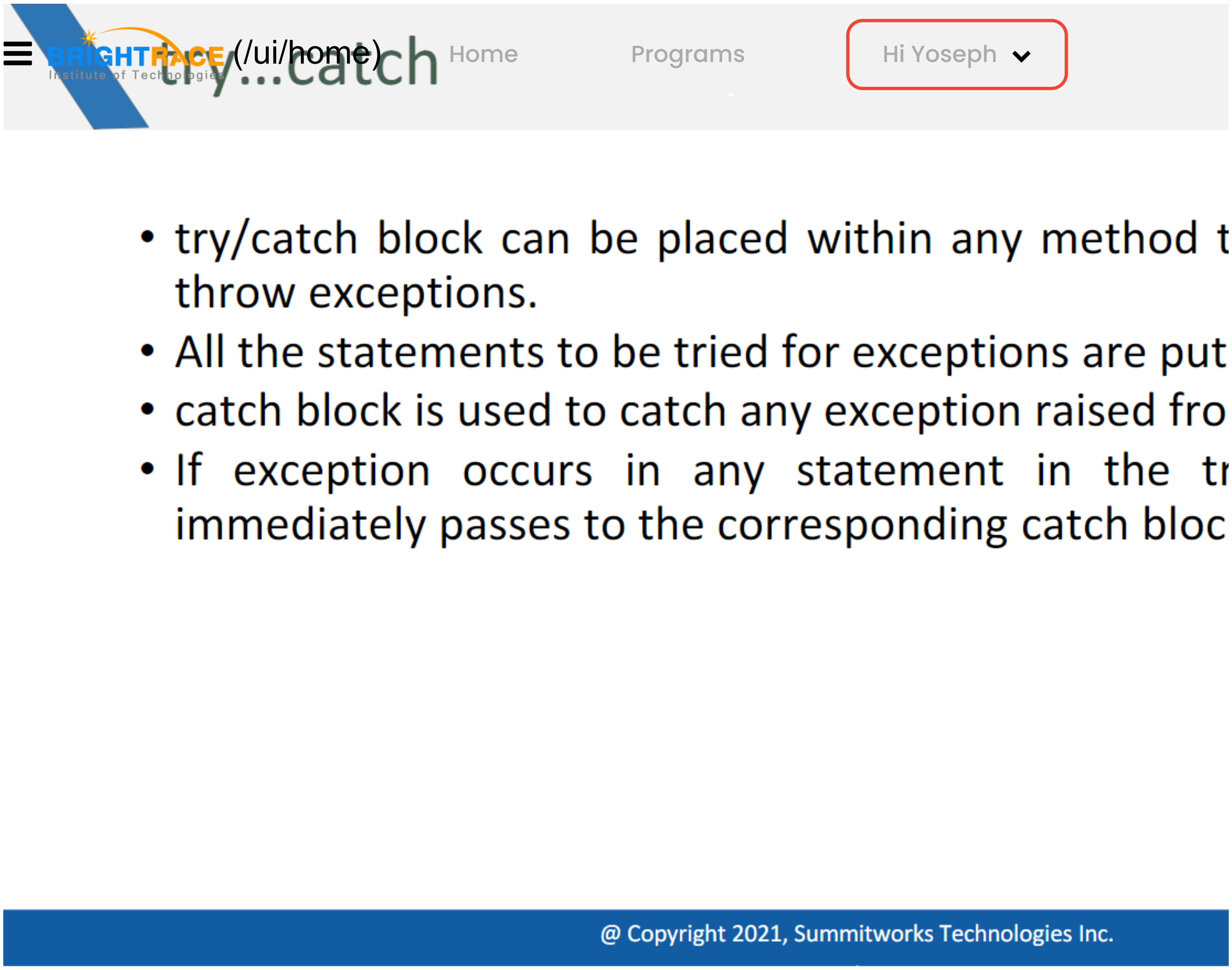
Exception Hierarchy



Exception Handling Techniques

- try..catch
- throw
- throws
- finally

(/ui/home) Home Programs Hi Yoseph



try...catch

- try/catch block can be placed within any method t
  throw exceptions.
- All the statements to be tried for exceptions are put
- catch block is used to catch any exception raised fro
- If exception occurs in any statement in the tr
  immediately passes to the corresponding catch bloc

Example

```
static void method2()
{
        System.out.println("IN Method 2, Calling Method 3");
        try{
                method3();
        }
        catch(Exception e)
        {
                System.out.println("Exception Handled");
        }
System.out.println("Returned from method 3");
}
```

Multiple catch clauses

```
static void method2()
{
        System.out.println("IN Method 2, Calling Method 3");
        try{
                method3(); }
        catch(ArithmeticException ae)
        {
                System.out.println ("Arithmetic Exception Handled: " +ae);
        }
        catch(Exception e)
        {
                System.out.println("Exception Handled");
        }
System.out.println("Returned from method 3");
}
```

Note: catch having super class types should be defined later than the catch c
The order is important.

Nested try..catch

- try{ …..//statements
- try{ …..//statements
- }
- catch(ArithmeticException ae){ . . . .}
- …// statements
- try{…//statements}
- catch(ArrayIndexOutOfBoundsException ie){}
- }
- catch(Exception e){…..}

throw keyword

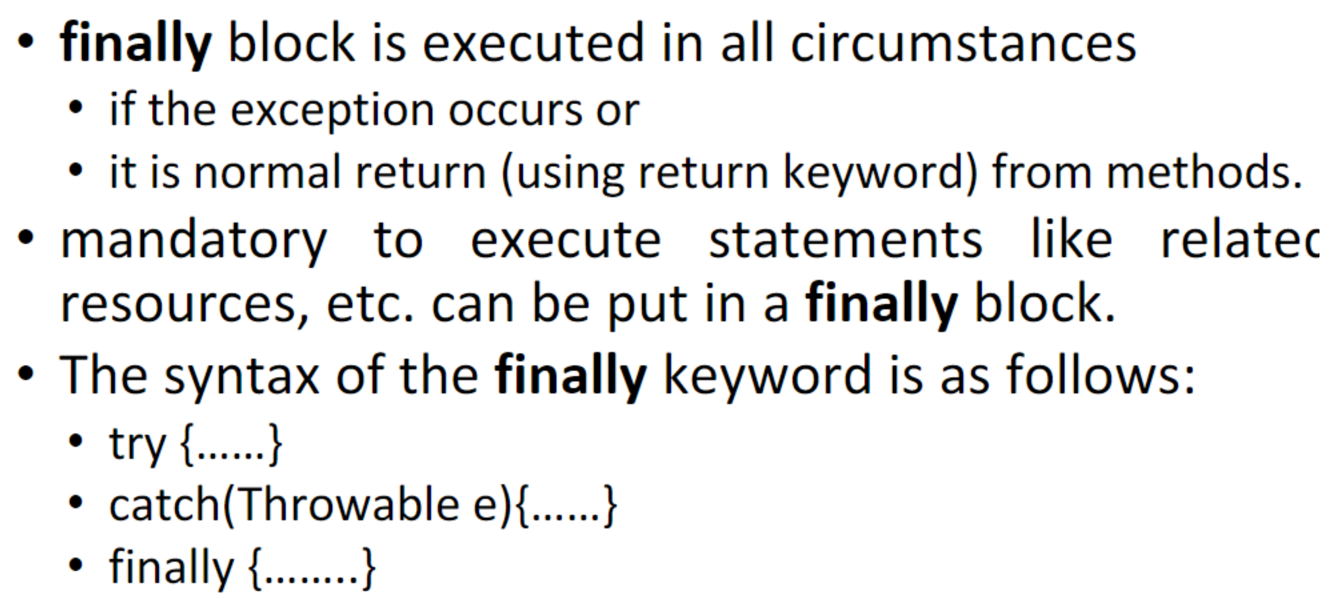
- used to explicitly throw an exception.
- useful when we want to throw a user-defined excep
- The syntax for *throw* keyword is as follows:
  - throw new ThrowableInstance;

  For example
  - throw new NullPointerException();

throws keyword

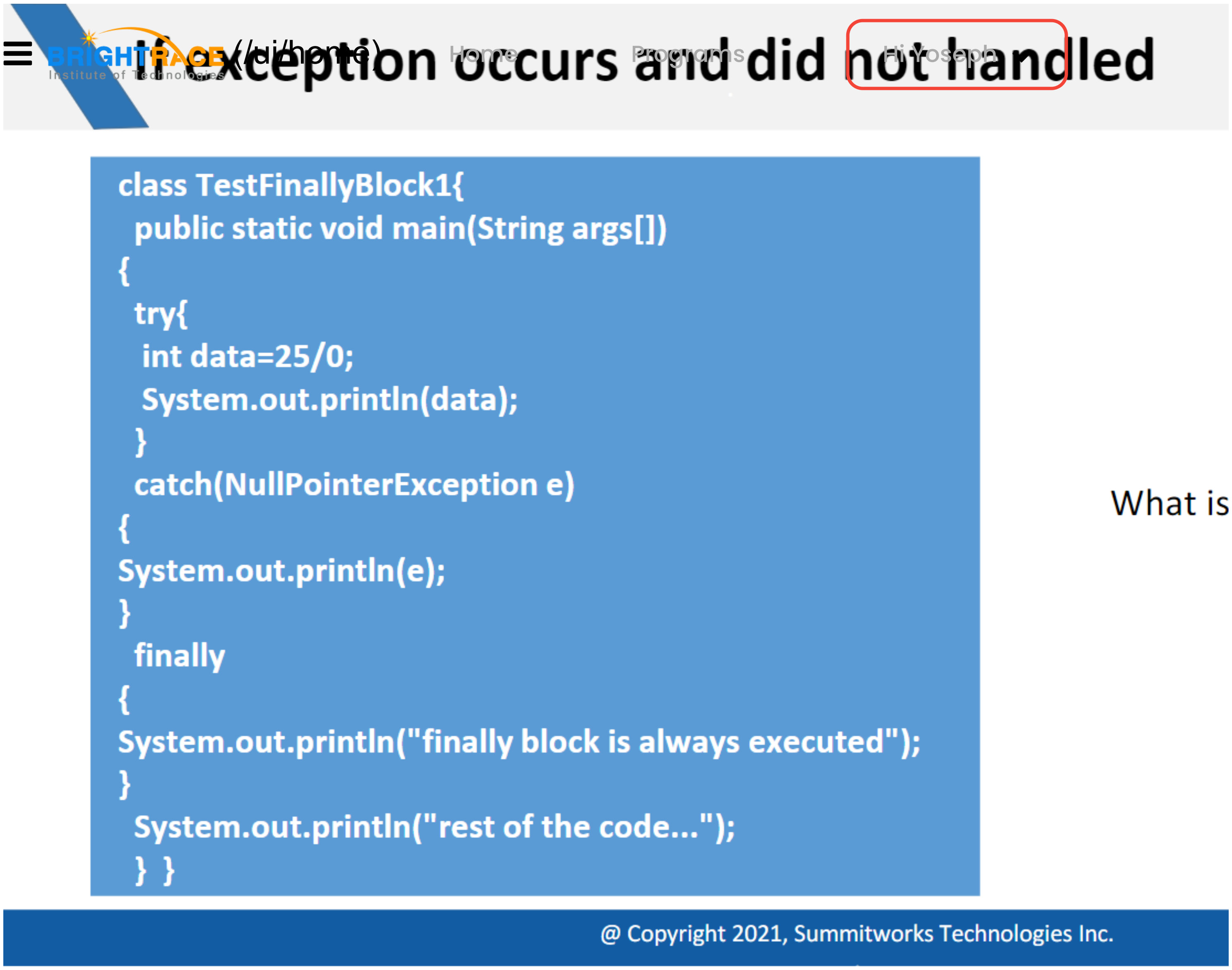
- is added to the method signature to let the caller l exception the called method can throw.
- responsibility of the caller to either handle the try...catch mechanism) or it can also pass th specifying throws clause in its method declaration).
- If all the methods in a program pass the exceptio (including main( )), then ultimately the exceptic default exception handler.

finally



- **finally** block is executed in all circumstances
  - if the exception occurs or
  - it is normal return (using return keyword) from methods.
- mandatory to execute statements like relatec resources, etc. can be put in a **finally** block.
- The syntax of the **finally** keyword is as follows:
  - try {……}
  - catch(Throwable e){……}
  - finally {……..}

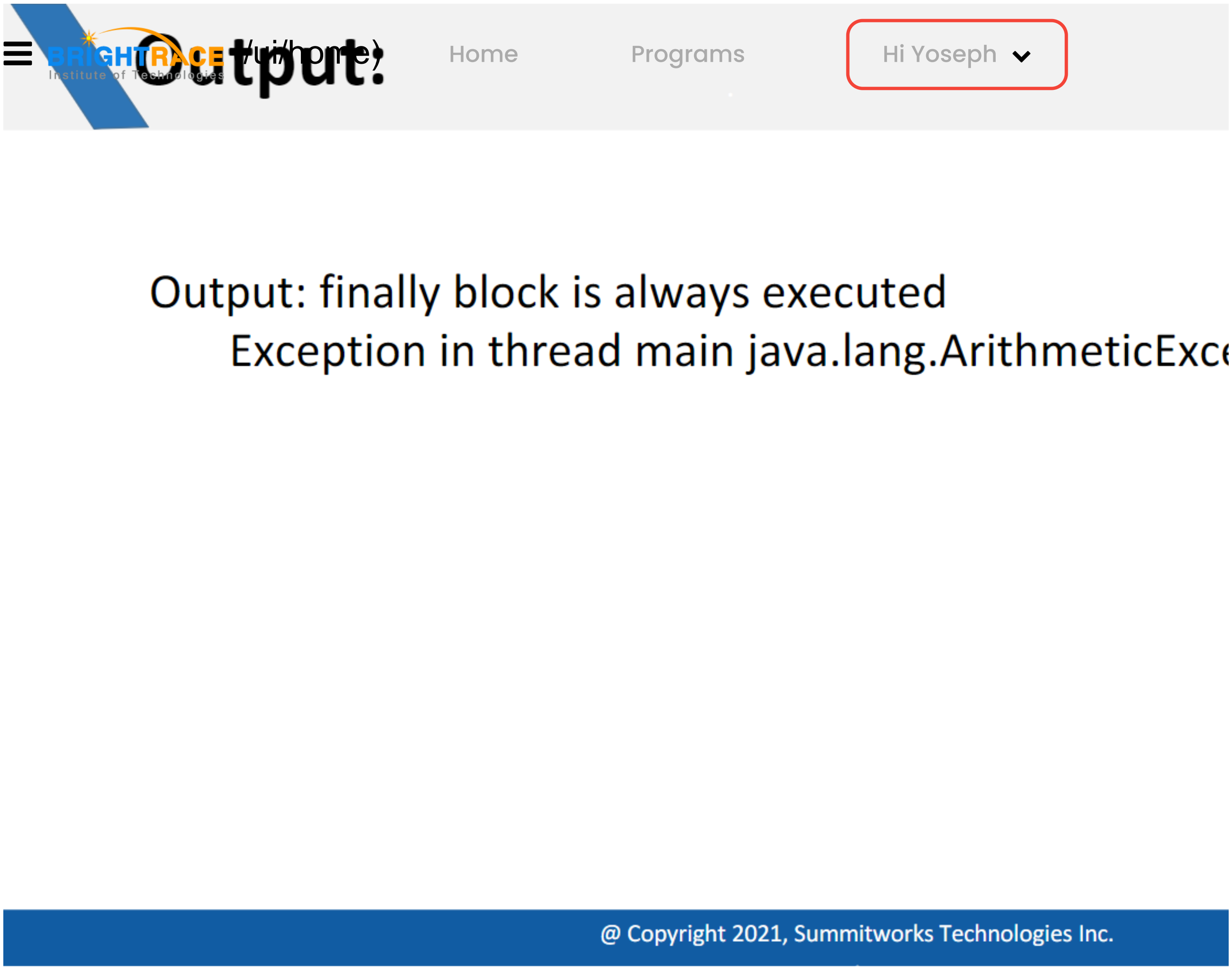
The finally block will not be executed if program exits(either by calling Sy: return, finally block will be executed



If exception occurs and did not handled

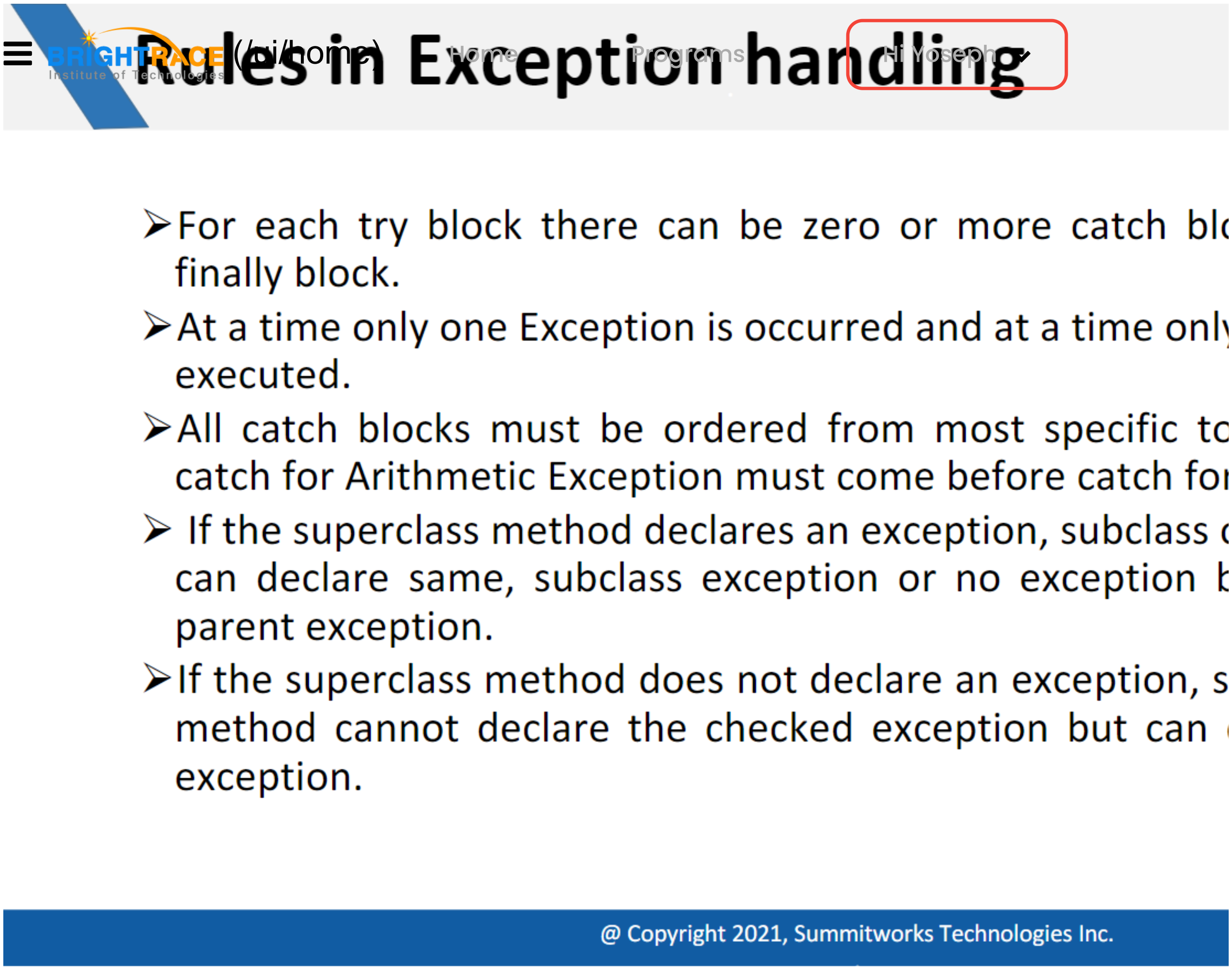
```
class TestFinallyBlock1{
  public static void main(String args[])
{
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(NullPointerException e)
{
System.out.println(e);
}
  finally
{
System.out.println("finally block is always executed");
}
  System.out.println("rest of the code...");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

What is



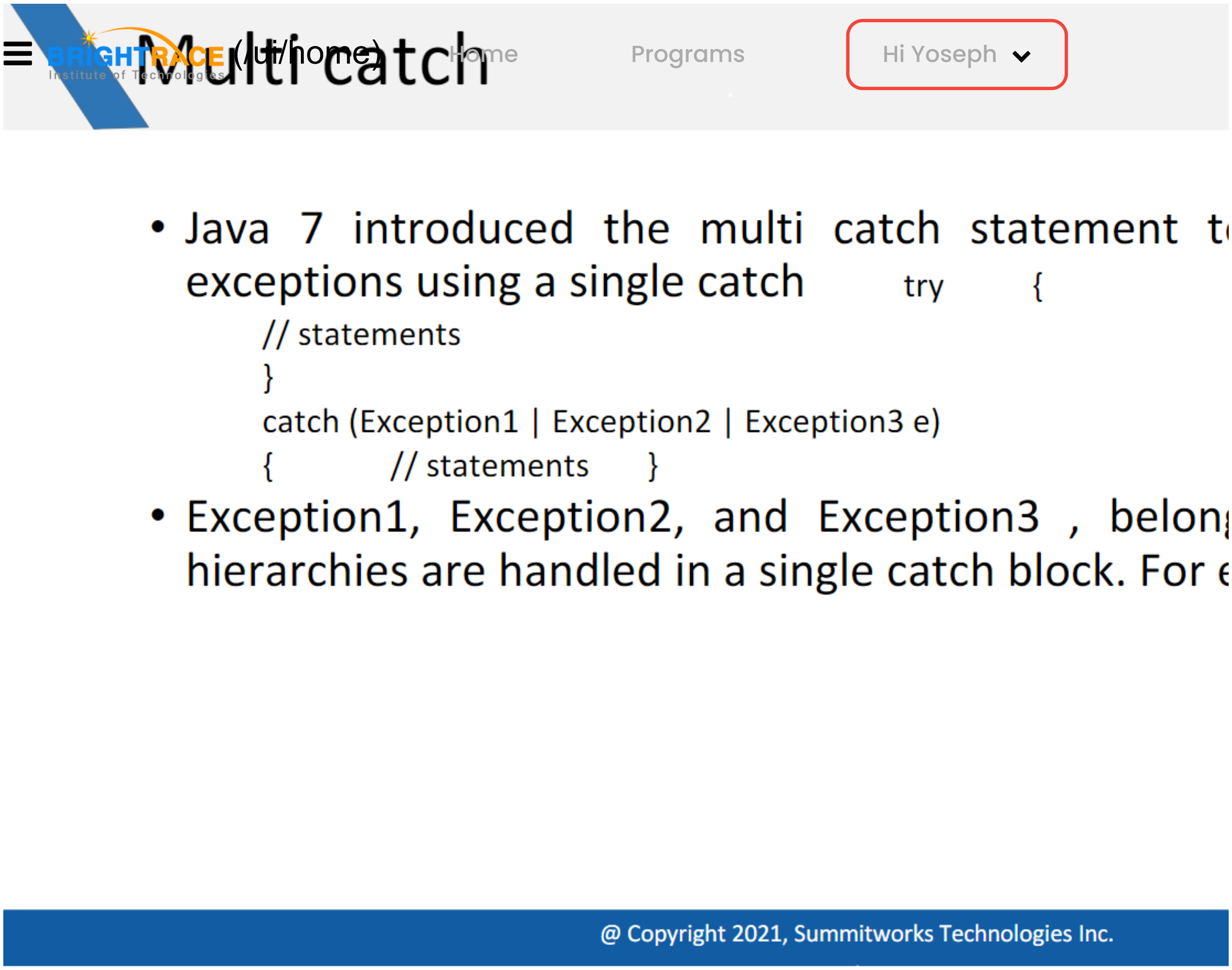
Output:

```
Output: finally block is always executed
        Exception in thread main java.lang.ArithmeticExce
```



Rules in Exception handling

- For each try block there can be zero or more catch blo finally block.
- At a time only one Exception is occurred and at a time only executed.
- All catch blocks must be ordered from most specific to catch for Arithmetic Exception must come before catch for
- If the superclass method declares an exception, subclass can declare same, subclass exception or no exception b parent exception.
- If the superclass method does not declare an exception, s method cannot declare the checked exception but can exception.

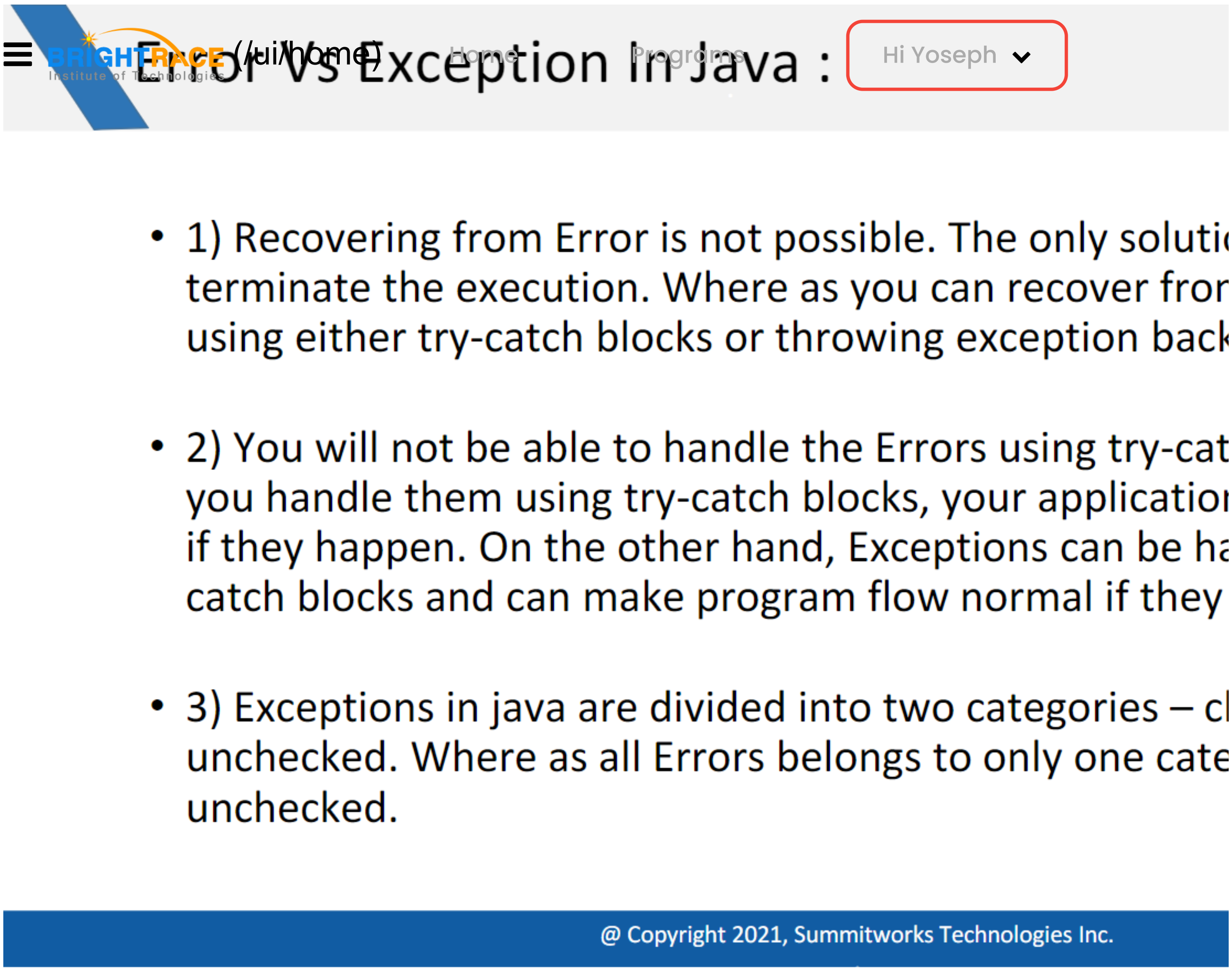


Multi catch

- Java 7 introduced the multi catch statement t
  exceptions using a single catch

```
try {
    // statements
}
catch (Exception1 | Exception2 | Exception3 e)
{        // statements    }
```

- Exception1, Exception2, and Exception3 , belon
  hierarchies are handled in a single catch block. For e

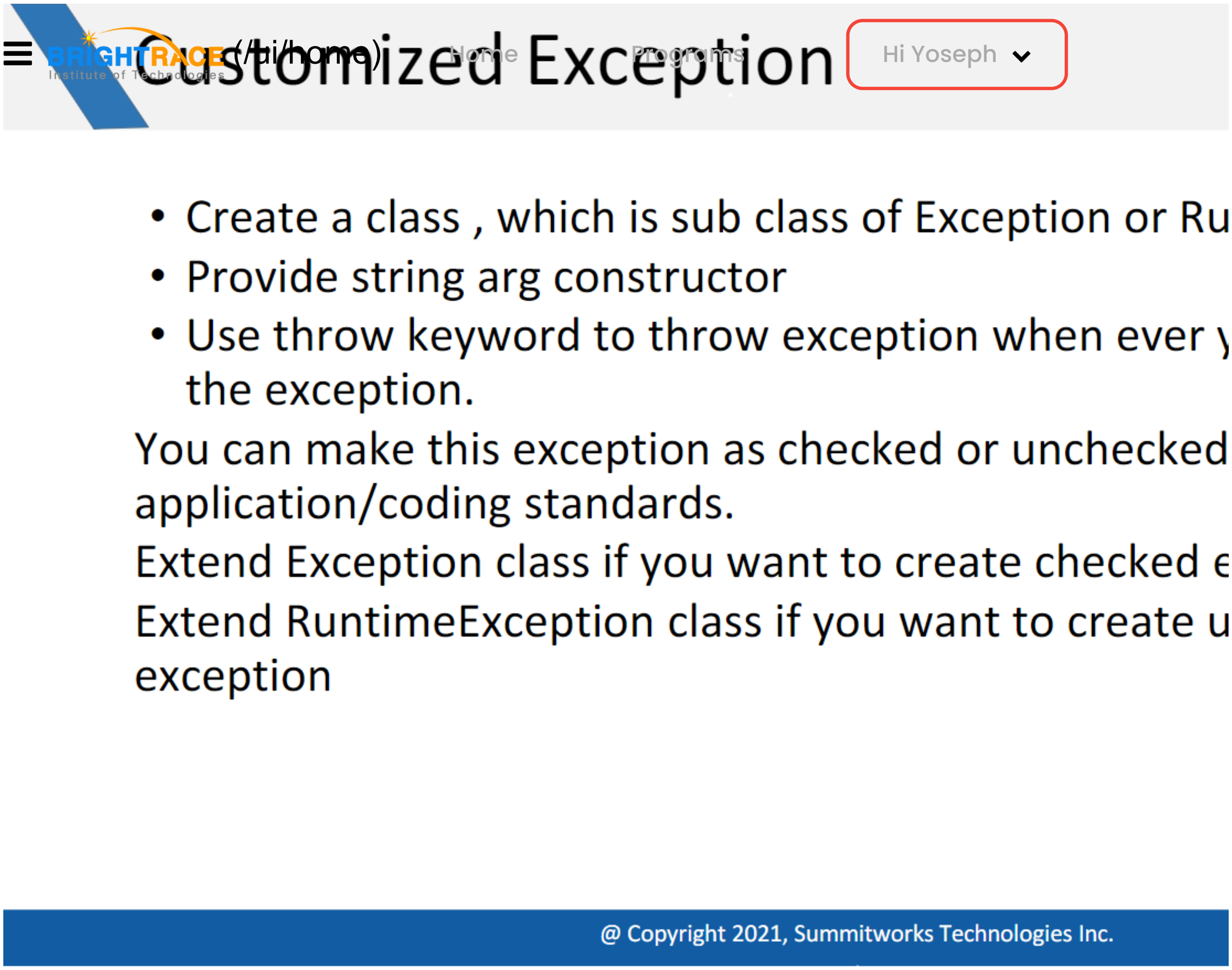


Error Vs Exception in Java :

- 1) Recovering from Error is not possible. The only soluti
terminate the execution. Where as you can recover fror
using either try-catch blocks or throwing exception back

- 2) You will not be able to handle the Errors using try-cat
you handle them using try-catch blocks, your applicatio
if they happen. On the other hand, Exceptions can be ha
catch blocks and can make program flow normal if they

- 3) Exceptions in java are divided into two categories – cl
unchecked. Where as all Errors belongs to only one cate
unchecked.



Customized Exception

- Create a class , which is sub class of Exception or Ru
- Provide string arg constructor
- Use throw keyword to throw exception when ever y the exception.

You can make this exception as checked or unchecked application/coding standards.

Extend Exception class if you want to create checked e

Extend RuntimeException class if you want to create u exception

Queries?