SNS COLLEGE OF TECHNOLOGY

(an autonomous institution)

Coimbatore - 35

19CST102 - OBJECT ORIENTED PROGRAMMING

I YEAR / II SEMESTER

UNIT IV – MULTITHREADING IN JAVA

TOPIC : Exception handling with try,catch,finally

Presented by:
ROSHAN A
(713522CS129)
SHIVAHARIHARAN N
(713522CS146)
SUDHARSANRAJA S S
(713522CS157)
VIJAYARAGHAVAN T
(713522CS168)

Guided by :
Mr.Selvakumar. N
AP/CSE

Exception Handling in Java

- The Exception Handling in Java is one of the powerful mechanism to **Handle the runtime errors** so that the normal flow of the application can be maintained.

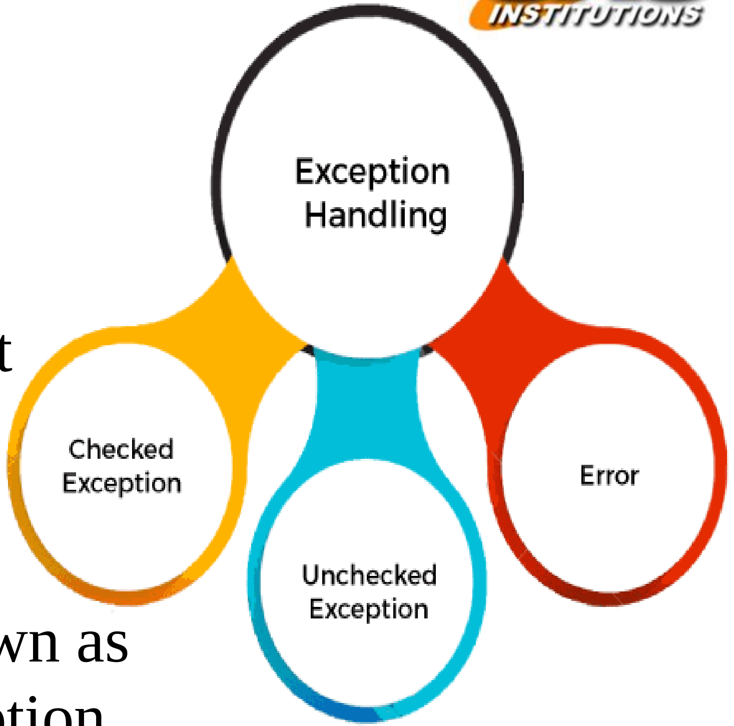
- When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (throw an error).

Types of Java Exceptions:

- There are mainly two types of exceptions: **checked** and **unchecked.** An **Error** is considered as the unchecked exception.

Checked Exception:

- The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time



Unchecked Exception:

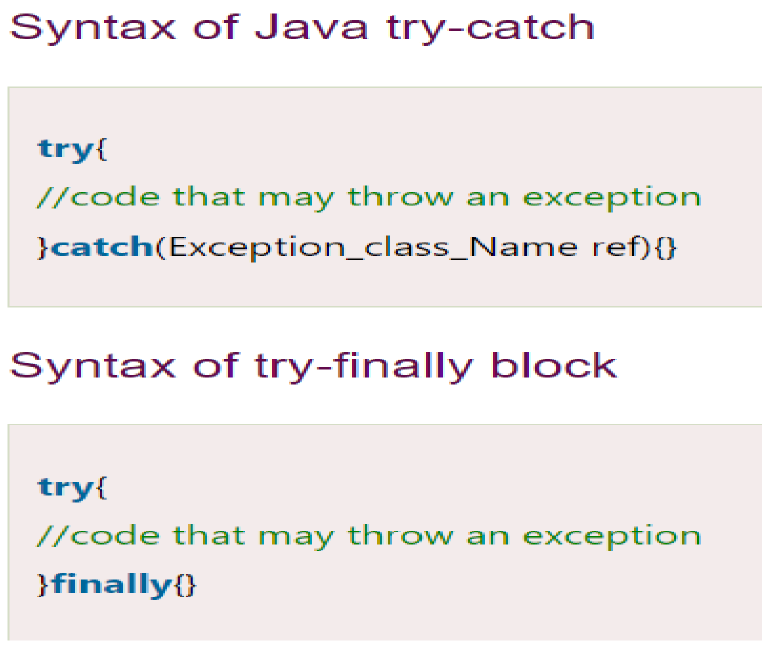
- The classes that inherit the RuntimeException are known as unchecked exceptions. For example, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

Error:

- Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords:

1. TRY
2. CATCH
3. FINALLY
4. THROW
5. THROWS



Syntax of Java try-catch

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

Syntax of try-finally block

```
try{
//code that may throw an exception
}finally{}
```

Java try-catch block :

Java try block

- Java try block is used to **enclose the code that might throw an exception**. It must be used within the method . If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. So, it is recommended not to keep the code in try block that will not throw an exception.

- Java try block must **be followed by either catch or finally block**.

Java catch block :

- Java catch block is used to handle the **Exception by declaring the type of exception within the parameter**. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type.

Problem without exception handling

Example 1

```
public class Main {
 public static void main(String[ ] args)
 {
  int[] myNumbers = {1, 2, 3};
 System.out.println(myNumbers[10]); // error!
 }}
```

Output:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10 at
Main.main(Main.java:4)
```

Solution by exception handling:

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

   try {      int[] myNumbers = {1, 2, 3};

     System.out.println(myNumbers[10]);    }
 catch (Exception e) {

      System.out.println("Something went wrong.");
   } }}
```

OUTPUT

```
    SOMETHING WENT  WRONG
```

Java finally block:

- It allows the programmer to avoid having cleanup code accidentally bypassed by a return , continue , or break
- The finally statement lets you **execute code**, after try...catch, **regardless of the result**
- The important statements to be printed can be placed in the finally block.

Example

```
public class Main {

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

      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
catch (Exception e) {

      System.out.println("Something went wrong.");

    }
      finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

output

```
Something went wrong.
The 'try catch' is finished.
```

THROW KEYWORD:

- The throw statement allows you to create a custom error.
- The throw statement is used together with an **exception type**. There are many exception types available in
- Java:
    - ArithmeticException,
    - FileNotFoundException,
    - ArrayIndexOutOfBoundsException,
    - SecurityException

```
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }
  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

OUTPUT:

```
Exception in thread "main" java.lang.ArithmeticException: Access denied - You must be at least 18 years old.
        at Main.checkAge(Main.java:4)
        at Main.main(Main.java:12)
```

Thank you!