SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Information Technology

19CS204 OBJECT ORIENTED PROGRAMMING

I YEAR /II SEMESTER

Topic – Exception Handling – Nested try, throw, throws, finally

Exception Handling – Nested Try

Nested try Statements

- The try statement can be nested. That is, a try statement can be inside the block of another try.
- Each time a try statement is entered, the context of that exception is pushed on the stack.
- If an inner try statement does not have a catch handler for a particular exception, the stack is unwound and the next try statement's catch handlers are inspected for a match.
- This continues until one of the catch statements succeeds, or until all of the nested try statements are exhausted.
- If no catch statement matches, then the Java run-time system will handle the exception.

Exception Handling – Nested Try

```
public class NestTry {
public static void main(String args[]) {
try {
int a = args.length;
/* If no command-line args are present,
the following statement will generate
a divide-by-zero exception. */
int b = 42 / a;
System.out.println("a = " + a);
try { // nested try block
/* If one command-line arg is used,
then a divide-by-zero exception
will be generated by the following code. */
if(a==1) a = a/(a-a); // division by zero
/* If two command-line args are used,
then generate an out-of-bounds exception. */
if(a==2) {
int c[] = { 1 };
c[42] = 99; // generate an out-of-bounds exception
}}
catch(ArrayIndexOutOfBoundsException e) {
System.out.println("Array index out-of-bounds: " + e);
}
} catch(ArithmeticException e) {
System.out.println("Divide by 0: " + e);
}
}
}
```

Exception Handling – throw

- So far, you have only been catching exceptions that are thrown by the Java run-time system.
- However, it is possible for your program to throw an exception explicitly, using the throw statement.
- The general form of throw is shown here:
  - *throw ThrowableInstance;*
- Here, ThrowableInstance must be an object of type Throwable or a subclass of Throwable.
- Primitive types, such as int or char, as well as non-Throwable classes, such as String and Object, cannot be used as exceptions.
- There are two ways you can obtain a Throwable object:
  - using a parameter in a catch clause or
  - creating one with the new operator.

Exception Handling – throw

Example – throw

```
public class Vote{
   static void validate(int age){
     if(age<18)
      throw new ArithmeticException("You are Not Eligible for Vote");
     else
      System.out.println("Welcome to vote");
   }
   public static void main(String args[]){
      validate(17);
      System.out.println("Thank you for voting");
  }
}
```

Exception Handling – throws

- If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception. You do this by including a throws clause in the method's declaration.

- Throws is necessary for all exceptions, except those of type Error or RuntimeException, or any of their subclasses.

- Usually, we don't need to handle unchecked exceptions. It's because unchecked exceptions occur due to programming errors. And, it is a good practice to correct them instead of handling them.

- All other exceptions that a method can throw must be declared in the throws clause. Checked exceptions.

- If they are not, a compile-time error will result. General form of a method declaration that includes a throws clause:

```
type method-name(parameter-list) throws exception-list
{
// body of method
}
```

- Here, exception-list is a comma-separated list of the exceptions that a method can throw.

Exception Handling – throws

Example

```
class ThrowsDemo {
static void throwOne() {
System.out.println("Inside throwOne.");
throw new IllegalAccessException("demo");
}
public static void main(String args[]) {
throwOne();
}
}
```

```
class ThrowsDemo {
static void throwOne() throws IllegalAccessException {
System.out.println("Inside throwOne.");
throw new IllegalAccessException("demo");
}
public static void main(String args[]) {
try {
throwOne();
} catch (IllegalAccessException e) {
System.out.println("Caught " + e);
}
}
}
```

Exception Handling – finally

- A finally block contains all the crucial statements that must be executed whether exception occurs or not.
- The statements present in this block will always execute regardless of whether exception occurs in try block.
- Java finally block follows try or catch block

Syntax of Finally block

```
try {
//Statements that may cause an exception
}
catch {
 //Handling exception
}
finally {
 //Statements to be executed
}
```

Exception Handling – finally

Example

```
public class finallyexample{
  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...");
  }
}
```

Exception Handling – finally

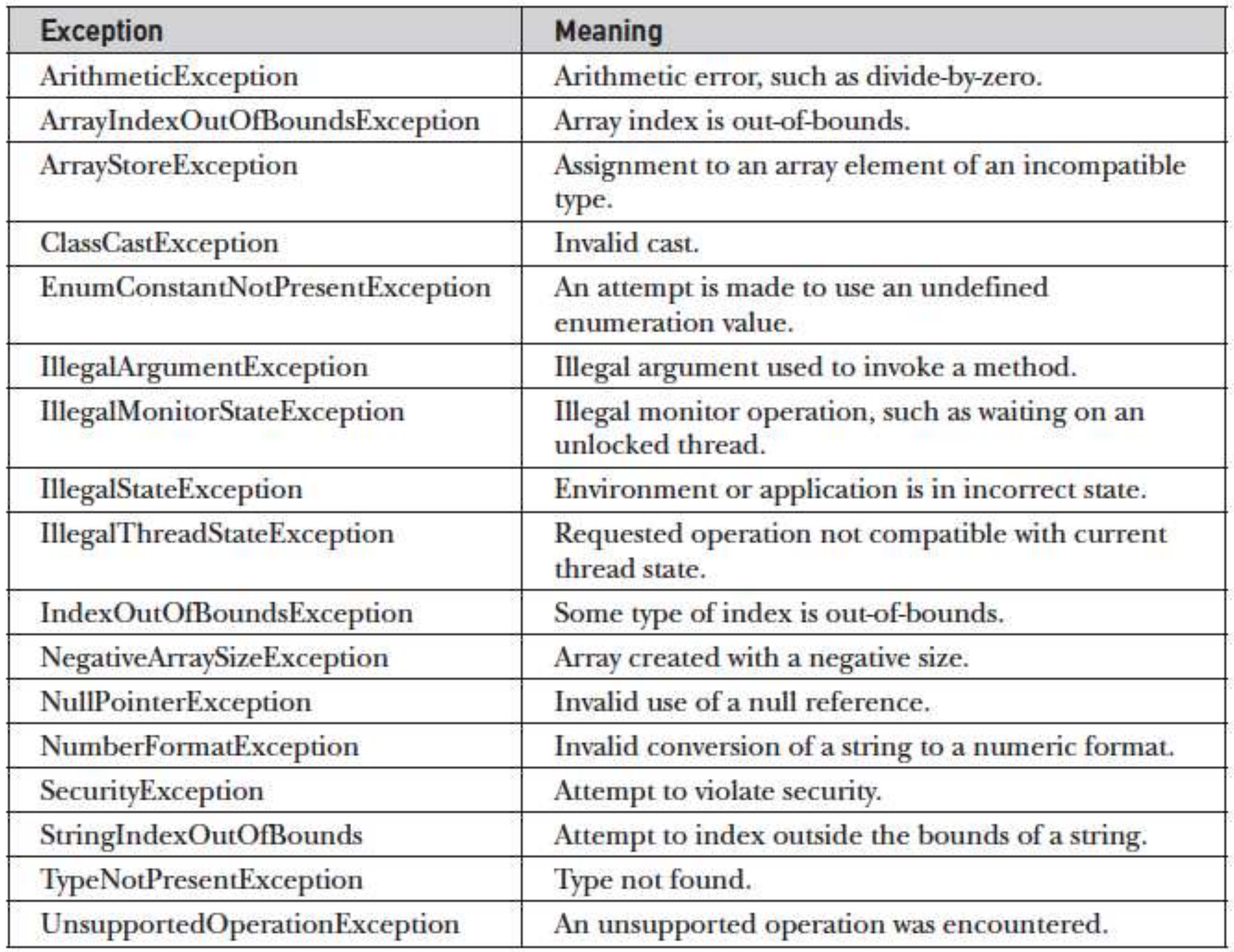
Example

```
class JavaFinally
{
   public static void main(String args[])
   {
      System.out.println(JavaFinally.myMethod());
   }
   public static int myMethod()
   {
      try {
        return 152;
      }
      finally {
        System.out.println("This is Finally block");
        System.out.println("Finally block ran even after return statement");
      }
   }
}
```

Exception Handling – finally

Java – Built-in Exceptions

Built in Exception of two types. Unchecked and checked exceptions. RuntimeException are called checked exceptions. Exceptions under RuntimeException subclass includes,

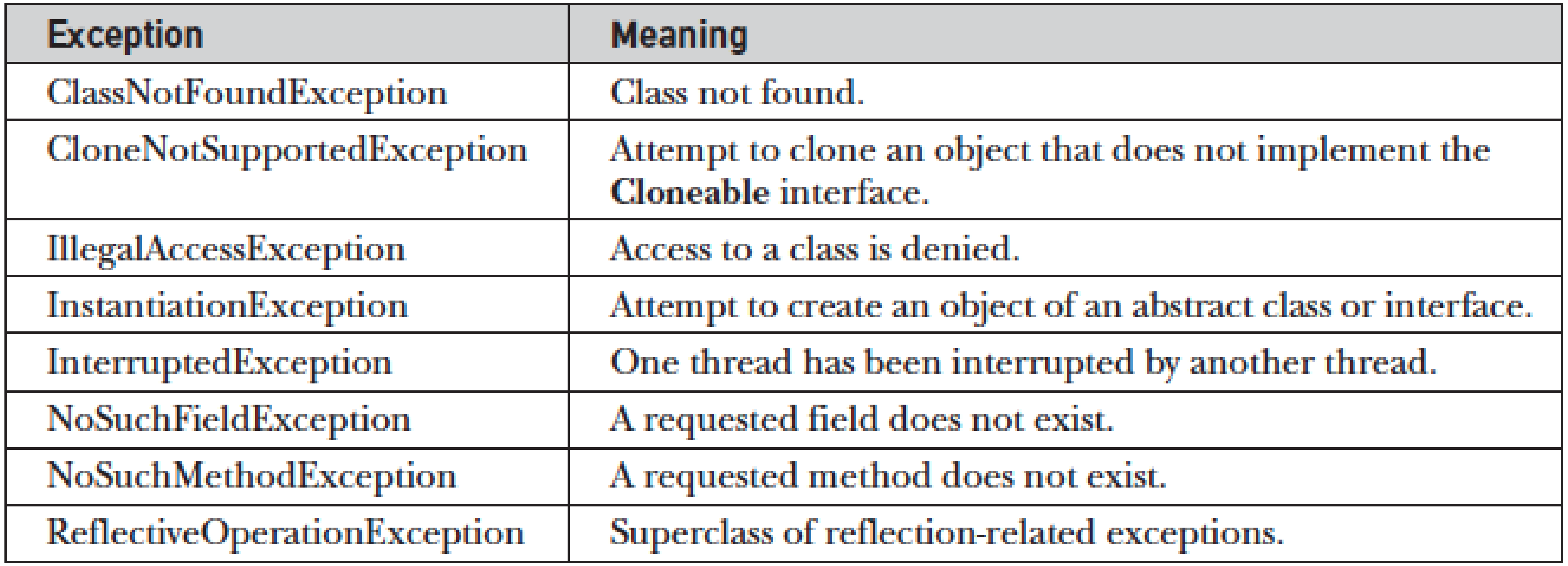


Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

Exception Handling – finally

Java – Built-in Exceptions

Checked exceptions includes exception that must be included in a method’s throws list if that method can generate one of these exceptions and does not handle it itself.



Exception	Meaning
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the **Cloneable** interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.
ReflectiveOperationException	Superclass of reflection-related exceptions.

Exception Handling – Creating Own Exceptions

User Defined or Custom Exceptions

- In java we can create our own exception class and throw that exception using throw keyword. These exceptions are known as **user-defined** or **custom** exceptions.
- Java custom exceptions are used to customize the exception according to user need.
- By the help of custom exception, you can have your own exception and message.
- Custom exceptions which are basically derived classes of Exception

Exception Handling – Creating Own Exceptions

Example

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 }  }

public class TestCustomException1{
 static void validate(int age)throws InvalidAgeException{
     if(age<18)
      throw new InvalidAgeException("not valid");
     else
      System.out.println("welcome to vote");
   }
public static void main(String args[]){
      try{
      validate(13);
      }catch(Exception m){System.out.println("Exception occured: "+m);}
      System.out.println("rest of the code...");   }
}
```

THANK YOU