Exception Handling

Exceptions

- An ***exception*** is an object that describes an **unusual or erroneous** situation.
- Exceptions are **thrown** by a program, and may be caught and handled by another part of the program.
- A program can be separated into a **normal execution flow** and an **exception execution flow.**
- An ***error*** is also represented as an object in Java, but usually represents a **unrecoverable situation** and should not be caught.

Exception Handling

- Java has a predefined set of exceptions and errors that can occur during execution.
- A program can deal with an exception in one of three ways:
  - ignore it
  - handle it where it occurs
  - handle it at another place in the program
- The manner in which an exception is processed is an important design consideration.

Using try and catch

- Example:

```
try{
        // code that may c
}
catch(Exception e){
        // code when exception occurred
}
```

Exception is the superclass of all the exception that may occur in Java

- Multiple catch:

```
try{
        // code that may cause exception
}
catch(ArithmeticException ae){
        // code when arithmetic exception occurred
}
catch(ArrayIndexOutOfBoundsException aiobe){
        // when array index out of bound exception occurred
}
```

Nested try statements

```
try
{
    try
    {
        // code that may cause array index out of bound exception
    }
    catch(ArrayIndexOutOfBoundsException aiobe)
    {
        // code when array index out of bound exception occured
    }
    // other code that may cause arithmetic exception
}
catch(ArithmeticException ae)
{
        // code when arithmetic exception occurred
}
```

throw statement

- 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.

Throw (Example)

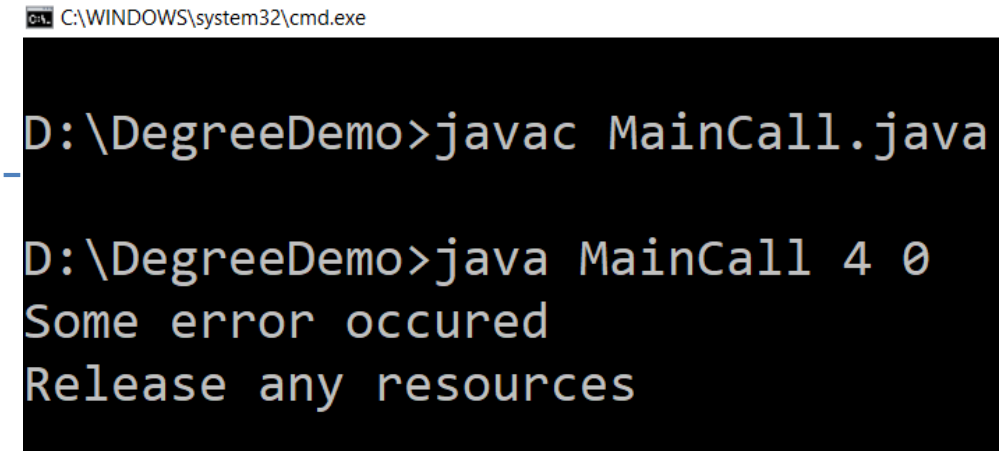
```
public class DemoException {

    public static void main(String[] args) {

        try {
                // Your Code Here
                throw new Exception("Custom Exception");
        } catch (Exception e) {
                System.out.println(e.getMessage());
        }

    }

}
```

The finally statement

- The purpose of the **finally** statement will allow the execution of a segment of code regardless if the try statement throws an exception or executes successfully
- The advantage of the **finally** statement is the ability to clean up and release resources that are utilized in the **try** segment of code that might not be released in cases where an exception has occurred.

The finally statement (Example)

```
public class MainCall {
    public static void main(String args[]) {
        int a,b;
        double c;
        a = Integer.parseInt(args[0]);
        b = Integer.parseInt(args[1]);
        try {
            c = a/b;
            System.out.println(c);
        }
        catch(Exception e) {
            System.out.println("Some error occurred");
        }
    }
}
```



throws statement

- A **throws** statement lists the types of exceptions that a method might throw.
- This is necessary for all exceptions, except those of type **Error** or **RuntimeException**, or any of their subclasses.
- All other exceptions that a method can throw must be declared in the throws clause. If they are not, a compile-time error will result.

throws statement (Cont.)

- This is the 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.*
- Example :

```
void myMethod() throws ArithmeticException, NullPointerException
{
    // code that may cause exception
}
```

Checked Exceptions

- An **exception** is either checked or unchecked.
- A **checked exception** either must be caught by a method, or must be listed in the throws clause of any method that may throw or propagate it.
- The **compiler** will issue an **error** if a **checked exception** is **not caught** or asserted in a **throws** clause

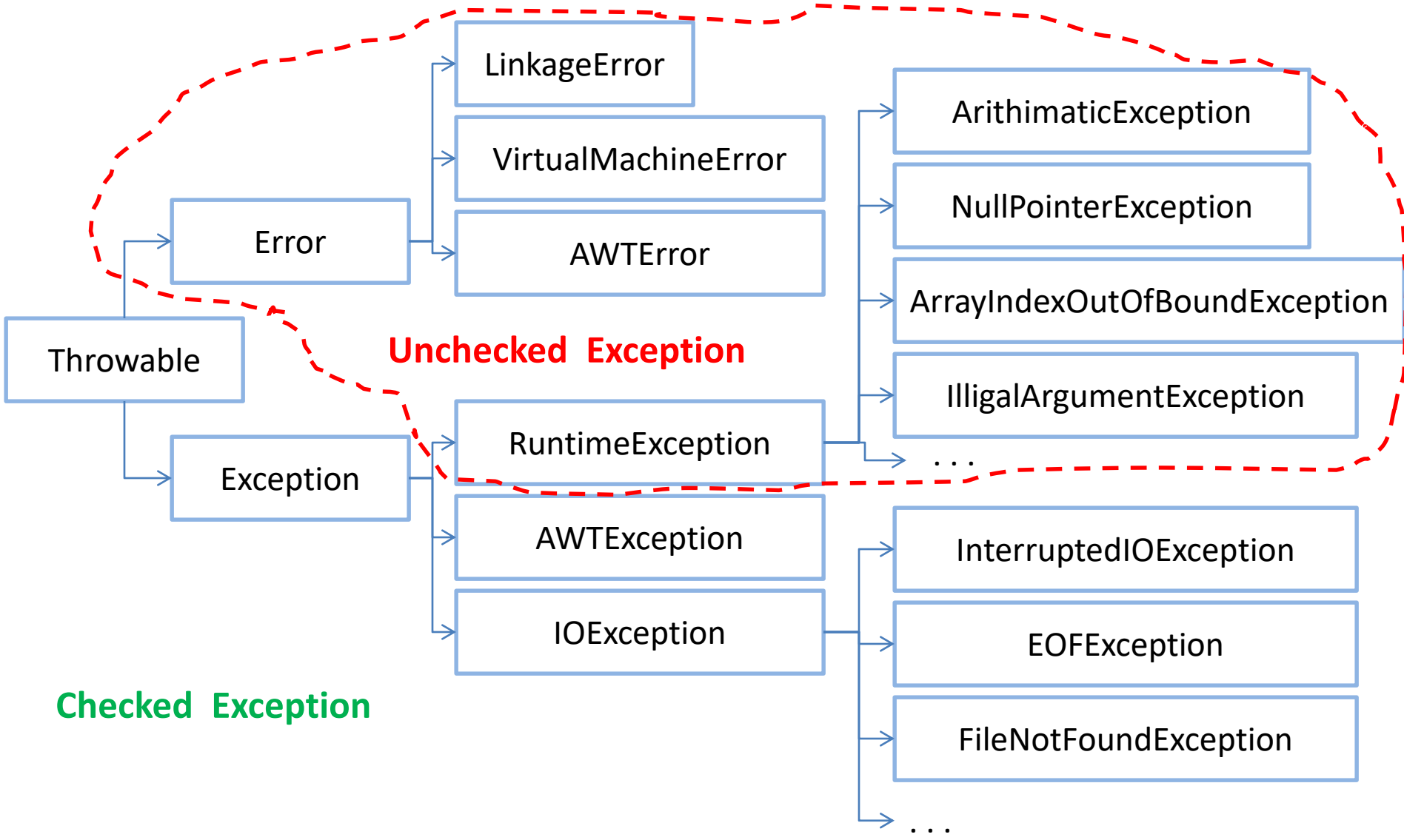
Unchecked Exceptions

- An **unchecked exception** does not require explicit handling, though it could be processed using try catch.
- The only unchecked exceptions in Java are objects of type **RuntimeException** or any of its descendants.

The Exception Class Hierarchy

- Classes that define exceptions are related by **inheritance**, forming an **exception class hierarchy**.
- All **error** and **exception** classes are descendents of the **Throwable** class
- The custom exception can be created by extending the **Exception** class or one of its descendants.

Exception Hierarchy



Java's built-in Exceptions

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ClassCastException	Invalid cast.
IllegalArgumentException	Illegal argument used to invoke a method.
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.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.

Unchecked Exceptions

Java's built-in Exceptions

Exception	Meaning
ClassNotFoundException	Class not found.
IOException	Input Output Exceptions
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.

Checked Exceptions

Create Your Own Exception

- Although Java's built-in exceptions handle most common errors, you will probably want to create your own exception types to handle situations specific to your applications.
- This is quite easy to do: just define a subclass of Exception (which is, of course, a subclass of Throwable).
- The Exception class does not define any methods of its own. It does inherit those methods provided by Throwable.
- Thus, all exceptions have methods that you create and defined by Throwable.

Create Your Own Exception (Cont.)

Method	Description
Throwable fillInStackTrace( )	Returns a Throwable object that contains a completed stack trace. This object can be rethrown.
Throwable getCause( )	Returns the exception that underlies the current exception. If there is no underlying exception, null is returned.
String getMessage( )	Returns a description of the exception.
StackTraceElement[ ] getStackTrace( )	Returns an array that contains the stack trace, one element at a time, as an array of StackTraceElement.
Throwable initCause(Throwable causeExc)	Associates causeExc with the invoking exception as a cause of the invoking exception. Returns a reference to the exception.
void printStackTrace( )	Displays the stack trace.
void printStackTrace(PrintStream stream)	Sends the stack trace to the specified stream.
void setStackTrace(StackTraceElement elements[ ])	Sets the stack trace to the elements passed in elements.
String toString( )	Returns a String object containing a description of the exception.

Custom Exception (Example)

```
// A Class that represents use-defined exception
class MyException extends Exception {
    public MyException(String s) {
        // Call constructor of parent (Exception)
        super(s);
    }
}
```

Custom Exception (Example) (Cont.)

```
class MainCall {
    static int currentBal = 5000;
    public static void main(String args[]) {
        try {
                int amount = Integer.parseInt(args[0]);
                withdraw(amount);
        } catch (Exception ex) {
            System.out.println("Caught");
            System.out.println(ex.getMessage());
        }
    }
    public static void withdraw(int amount) throws Exception
    {
        int newBalance = currentBal - amount;
        if(newBalance<1000) {
                throw new MyException("Darshan Exception");
        }
    }
}
```

```
D:\DegreeDemo>javac MainCall.java

D:\DegreeDemo>java MainCall 4500
Caught
Darshan Exception
```