Java

Agenda

- Exception Handling

Exception Handling

What Is an Exception?

- An **exception** is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.
- An exception can occur for many different reasons:
  - A user has entered invalid data.
  - A file that needs to be opened cannot be found.
  - A network connection has been lost in the middle of communications
  - the JVM has run out of memory.
- Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Types of Exception

- Checked Exception
- Unchecked Exception
  - Error
  - Runtime Exception

Checked

- Exceptional conditions that a well-written application should anticipate and recover from
- Example : File Reading process
  - An application prompts a user for an input file name, then opens the file by passing the name.
  - The user provides the name of an existing, readable file, and the execution of the application proceeds normally.
  - If the user supplies the name of a nonexistent file an exception occurs
- A well-written program will catch this exception and notify the user of the mistake

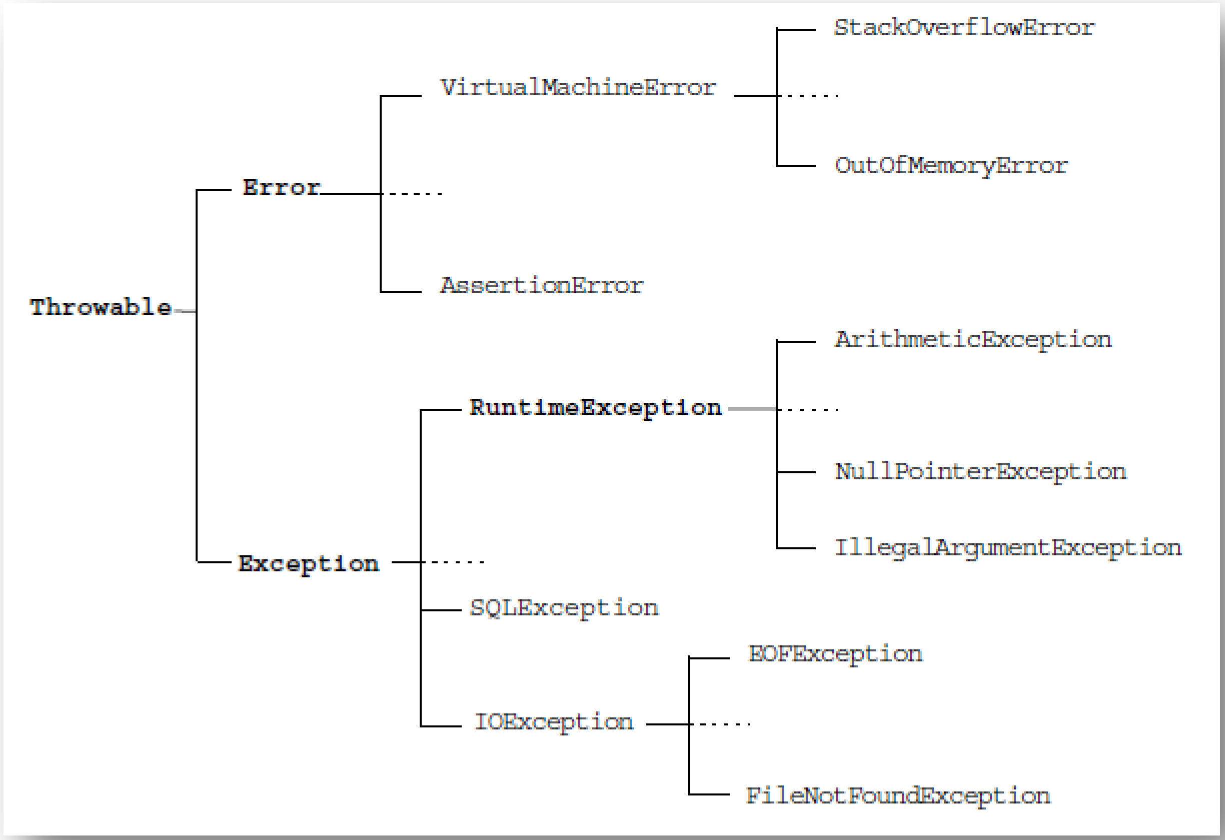
Error

- Exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from
- Example : File processing
  - An application successfully opens a file for input, but is unable to read the file because of a hardware or system malfunction.
  - The unsuccessful read will throw Error.
- An application might choose to catch this exception, in order to notify the user of the problem
- But it makes sense for the program to print a stack trace and exit.

Runtime Exception

- These are exceptional conditions that are internal to the application, and that the application usually cannot anticipate or recover from.
- These usually indicate programming bugs, such as logic errors or improper use of an packages.
- Example : File Processing
  - In the file reading application, if a logic error causes a null to be passed it will cause an Exception.
- The application can catch this exception, but it probably makes more sense to eliminate the bug that caused the exception to occur.

Exception Classes



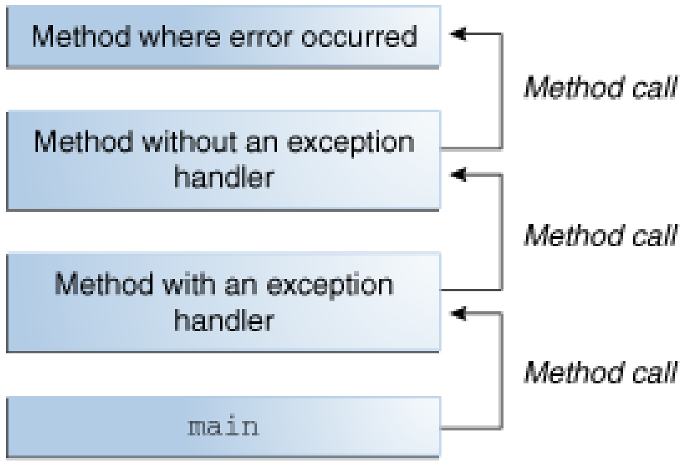
How are exceptions Handled???

Throwing an exception

- When an error occurs within a method, the method creates an object and hands it off to the runtime system – **exception object**
- The exception object contains information about the error
  - Type of exception
  - The state of the program when the error occurred
- Creating an exception object and handing it to the runtime system is called ***throwing*** *an exception*.

CALL STACK

- After a method throws an exception, the runtime system attempts to find something in the call stack to handle it.
- The list of methods that had been called to get to the method where the error occurred is known as the **call stack.**


Method where error occurred
Method call
Method without an exception handler
Method call
Method with an exception handler
Method call
main

Catching the exception

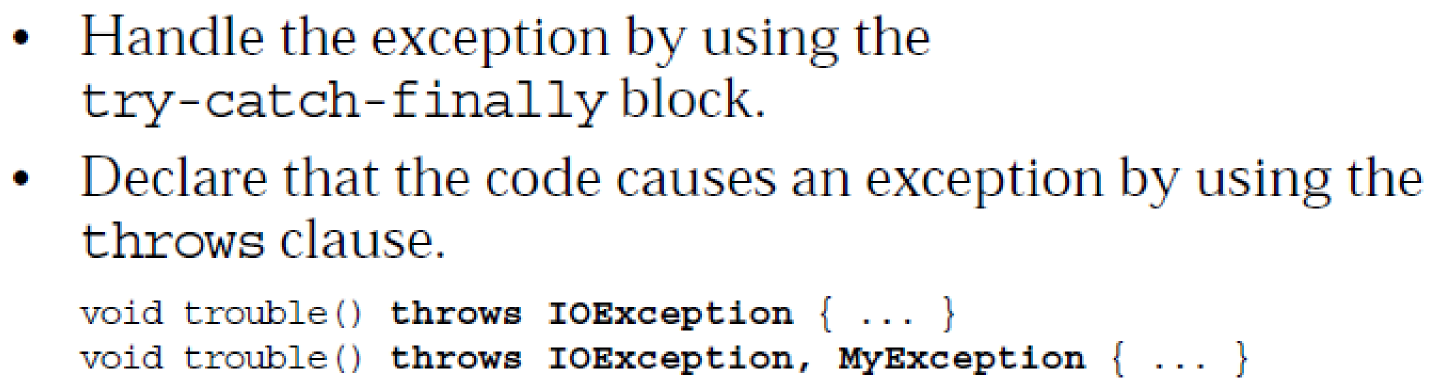
- The search begins with the method in which the error occurred
- Proceeds through the call stack in the reverse order in which the methods were called.
- The runtime system passes the exception object to the appropriate " **exception handler** - **a method that contains a block of code that can handle the exception "**
- The type of the exception object thrown should match the type that can be handled by the handler
- The exception handler chosen is said to *catch the exception*

Catch or Specify

Catch or Specify Requirement????

- The code that might throw certain exceptions must be enclosed by either of the following:
  1) **Try & catch** block that catches the exception
  2) A method that specifies that it can throw the exception. The method must provide a **throws** clause
- Code that fails to honor the Catch or Specify Requirement will not compile

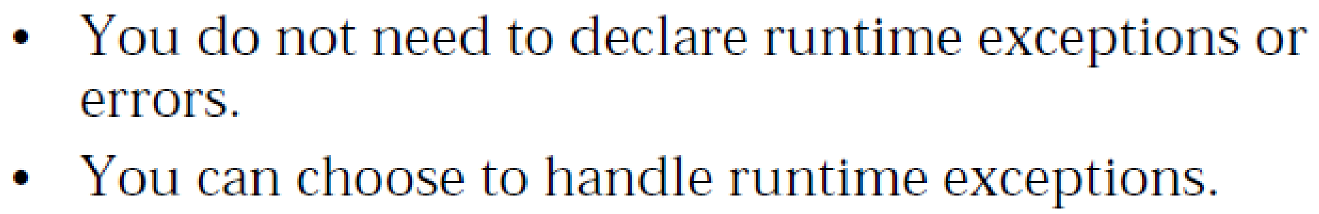
CATCH OR SPECIFY REQUIREMENT – CONTD…



- Handle the exception by using the `try-catch-finally` block.
- Declare that the code causes an exception by using the `throws` clause.

```
void trouble() throws IOException { ... }
void trouble() throws IOException, MyException { ... }
```

Other Principles



- You do not need to declare runtime exceptions or errors.
- You can choose to handle runtime exceptions.

Try Catch Finally

The try Block

- The first step in constructing an exception handler is to enclose the code that might throw an exception within a try block

```
try
{ code }
catch and finally blocks . . .
```

Catch Block

- If an exception occurs within the try block, that exception is handled by an exception handler associated with it.
- To associate an exception handler with a try block, a catch block is to be mentioned after it

```
try
 { }
catch (ExceptionType name)
 { }
```

Multiple Catch Blocks

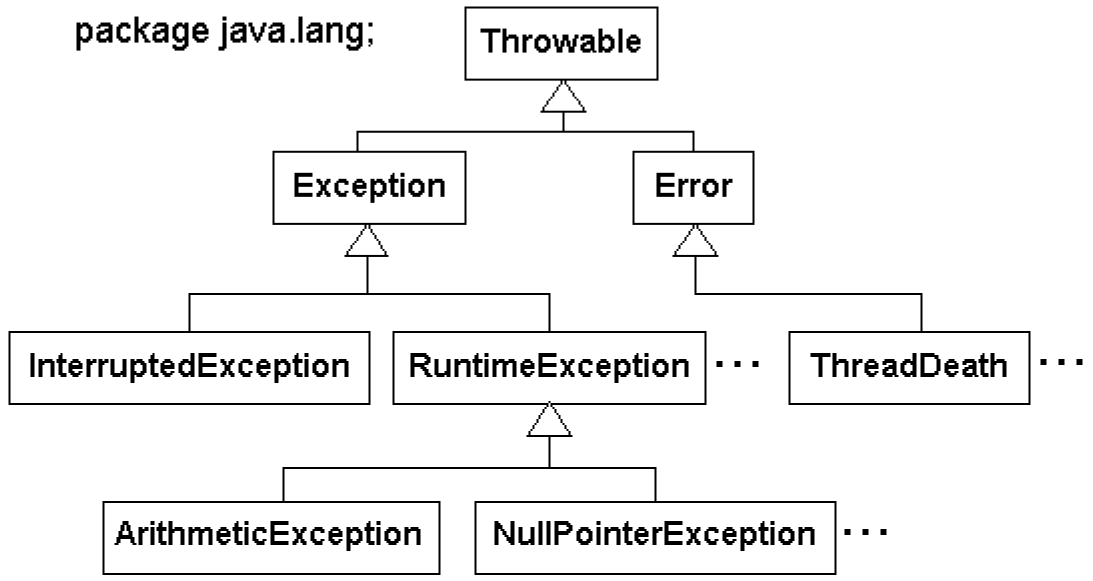
- Associate exception handlers with a try block by providing one or more catch blocks directly after the try block.
- No code can be between the end of the try block and the beginning of the first catch block.

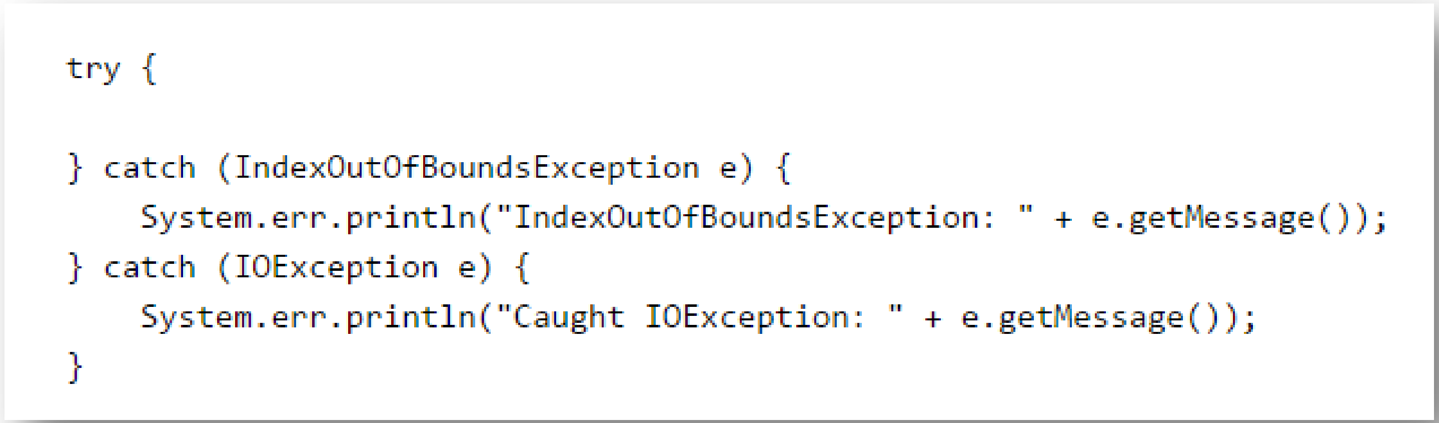
```
try
{    }
catch (ExceptionType name)
{    }
catch (ExceptionType name)
{    }
```

Throwable Class

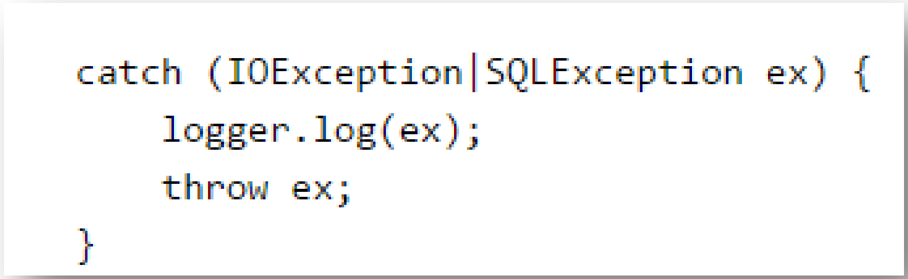
- Each catch block is an exception handler that handles the type of exception indicated by its argument.
- The argument type,*ExceptionType*, declares the type of exception that the handler can handle and must be the name of a class that inherits from theThrowable class.

Throwable Class





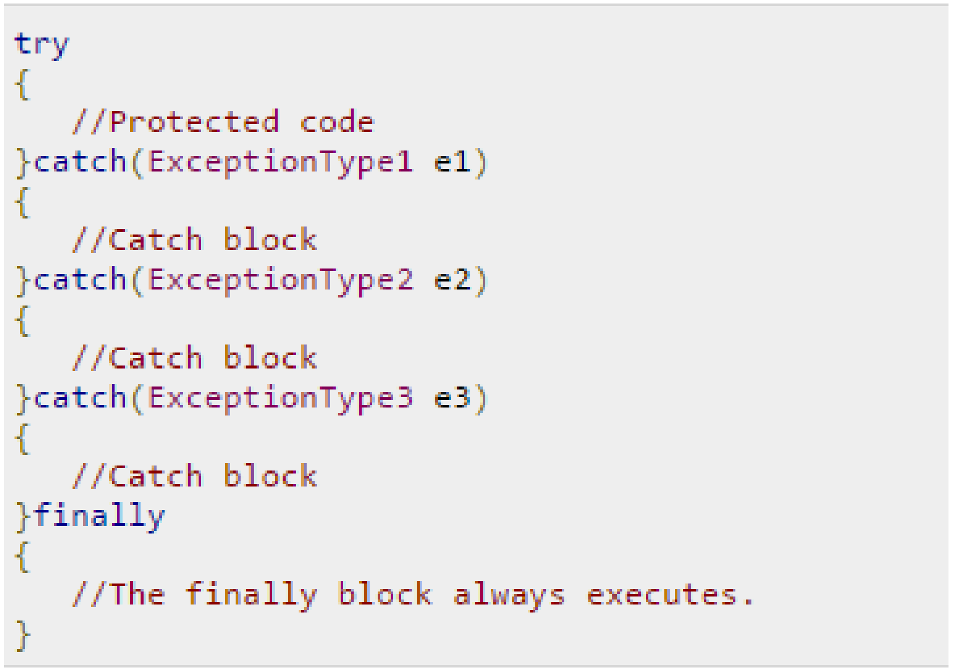
```
try {

} catch (IndexOutOfBoundsException e) {
    System.err.println("IndexOutOfBoundsException: " + e.getMessage());
} catch (IOException e) {
    System.err.println("Caught IOException: " + e.getMessage());
}
```



```
catch (IOException|SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

FINALLY BLOCK

- The finally block *always* executes when the try block exits.
- Finally block is executed even if an unexpected exception occurs
- Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.



```
try
{
   //Protected code
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}finally
{
   //The finally block always executes.
}
```

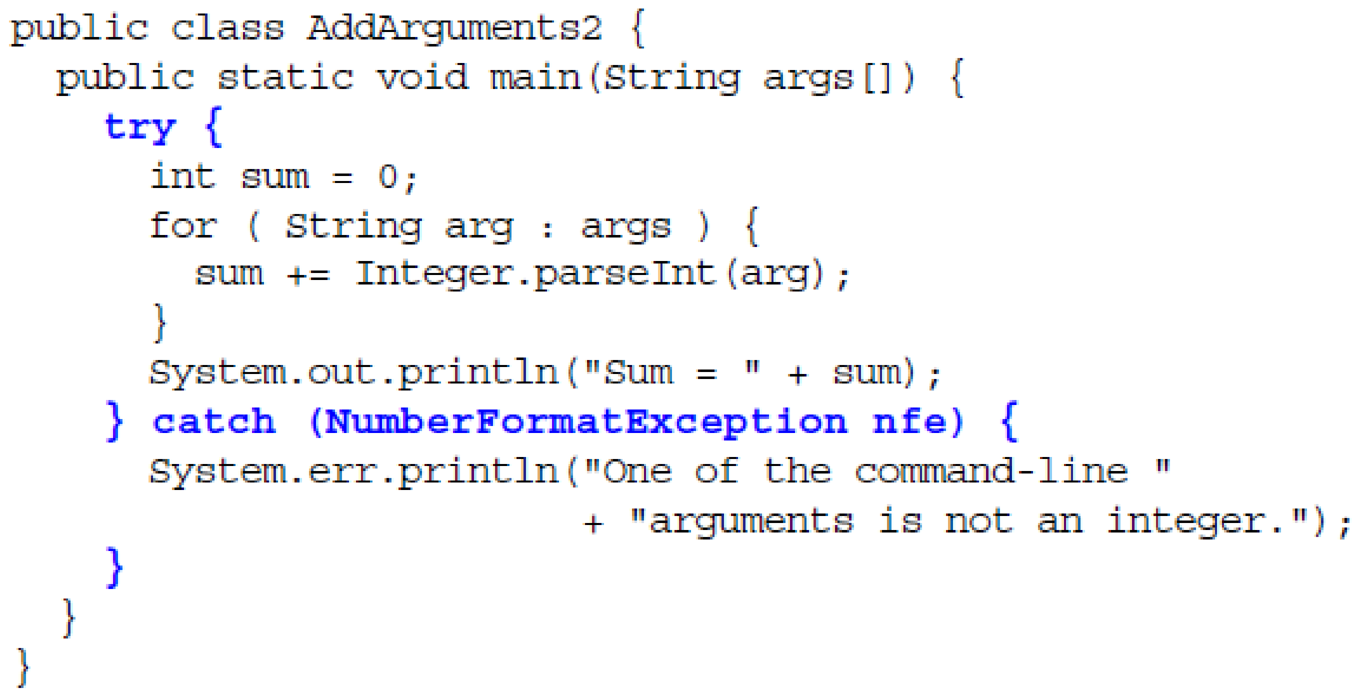
Command Line Argument

Command Line Argument

```
public class AddArguments
{
public static void main(String args[])
{
int sum = 0;
    for ( String arg : args )
    {
    sum += Integer.parseInt(arg);
    }
System.out.println("Sum = " + sum);
}
}
```

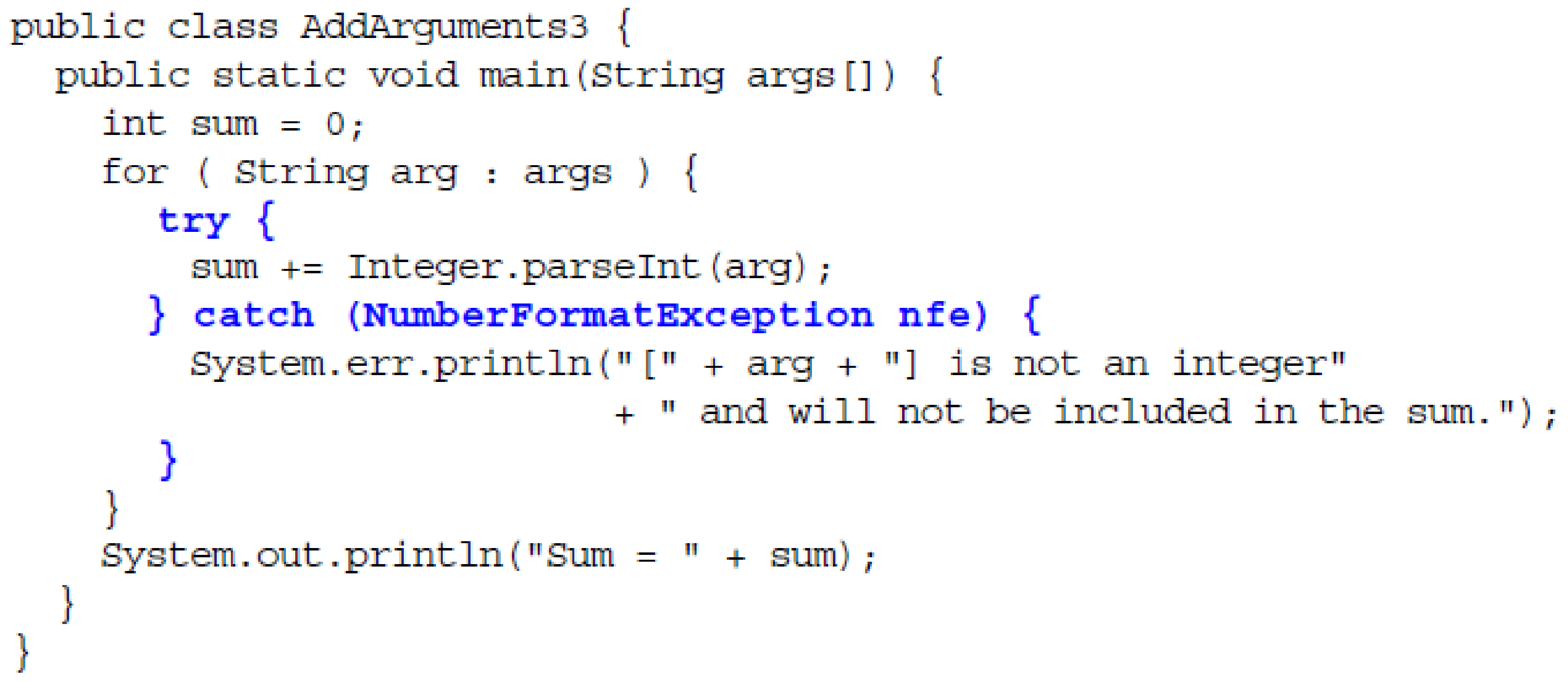
Exception Handling Examples

TRY..CATCH SAMPLE



```
public class AddArguments2 {
  public static void main(String args[]) {
    try {
      int sum = 0;
      for ( String arg : args ) {
        sum += Integer.parseInt(arg);
      }
      System.out.println("Sum = " + sum);
    } catch (NumberFormatException nfe) {
      System.err.println("One of the command-line "
                         + "arguments is not an integer.");
    }
  }
}
```

Granular Try..Catch Sample

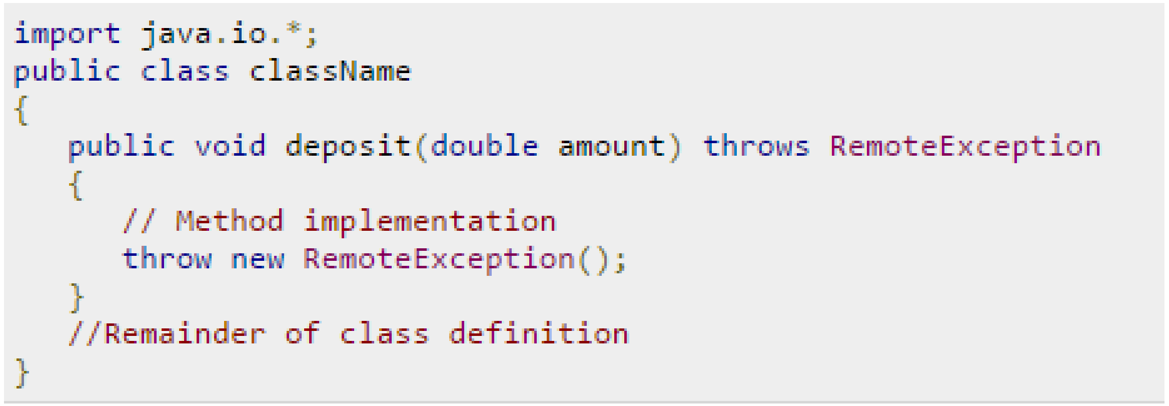


```
public class AddArguments3 {
  public static void main(String args[]) {
    int sum = 0;
    for ( String arg : args ) {
      try {
        sum += Integer.parseInt(arg);
      } catch (NumberFormatException nfe) {
        System.err.println("[" + arg + "] is not an integer"
                         + " and will not be included in the sum.");
      }
    }
    System.out.println("Sum = " + sum);
  }
}
```

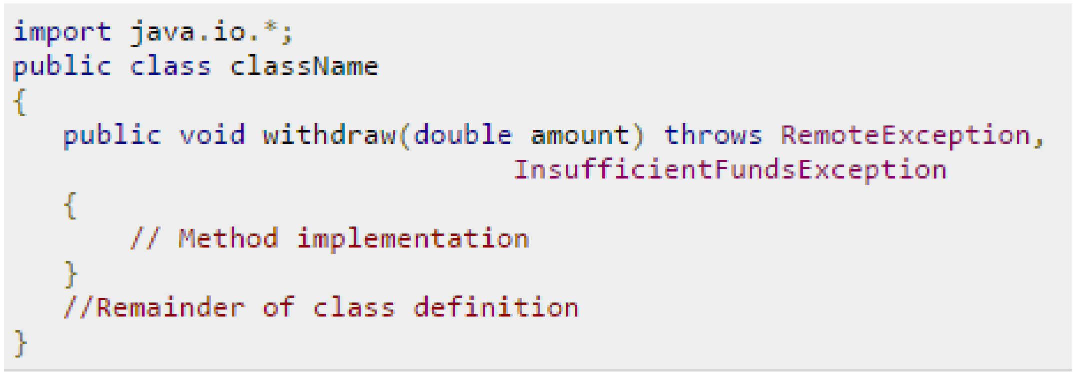
THE THROWS

- If a method does not handle a checked exception, the method must declare it using the **throws** keyword.
- The throws keyword appears at the end of a method's signature
- A method can declare that it throws more than one exception and the exceptions are declared in a list separated by commas.

Specifying the Exceptions Thrown by a Method



```java
import java.io.*;
public class className
{
   public void deposit(double amount) throws RemoteException
   {
      // Method implementation
      throw new RemoteException();
   }
   //Remainder of class definition
}
```



```java
import java.io.*;
public class className
{
   public void withdraw(double amount) throws RemoteException,
                              InsufficientFundsException
   {
       // Method implementation
   }
   //Remainder of class definition
}
```

Method Overriding & Exception Handling

The overriding method can throw:

- No exceptions
- One or more of the exceptions thrown by the overridden method
- One or more subclasses of the exceptions thrown by the overridden method

The overriding method cannot throw:

- Additional exceptions not thrown by the overridden method
- Superclasses of the exceptions thrown by the overridden method

```
public class TestA {
  public void methodA() throws IOException {
    // do some file manipulation
  }
}

public class TestB1 extends TestA {
  public void methodA() throws EOFException {
    // do some file manipulation
  }
}

public class TestB2 extends TestA {
  public void methodA() throws Exception { // WRONG
    // do some file manipulation
  }
}
```