Exceptions

CONTENTS

- **Objectives**
  - To explain exception handling in Java
- **Contents**
  - Exception handling syntax
  - Throwing exceptions
  - Understanding execution flow with exceptions
  - The try-with-resources statement
- **Hands on Labs**
  - Working with exceptions

Simple example of exception being thrown

- **Coding error**

```
public static void main(String[] args) {
  int[] ages = new int[7];
  ages[7] = 34;
}
```

Console output

```
Exception in thread "main"
   ArrayIndexOutOfBoundsException: 7
```

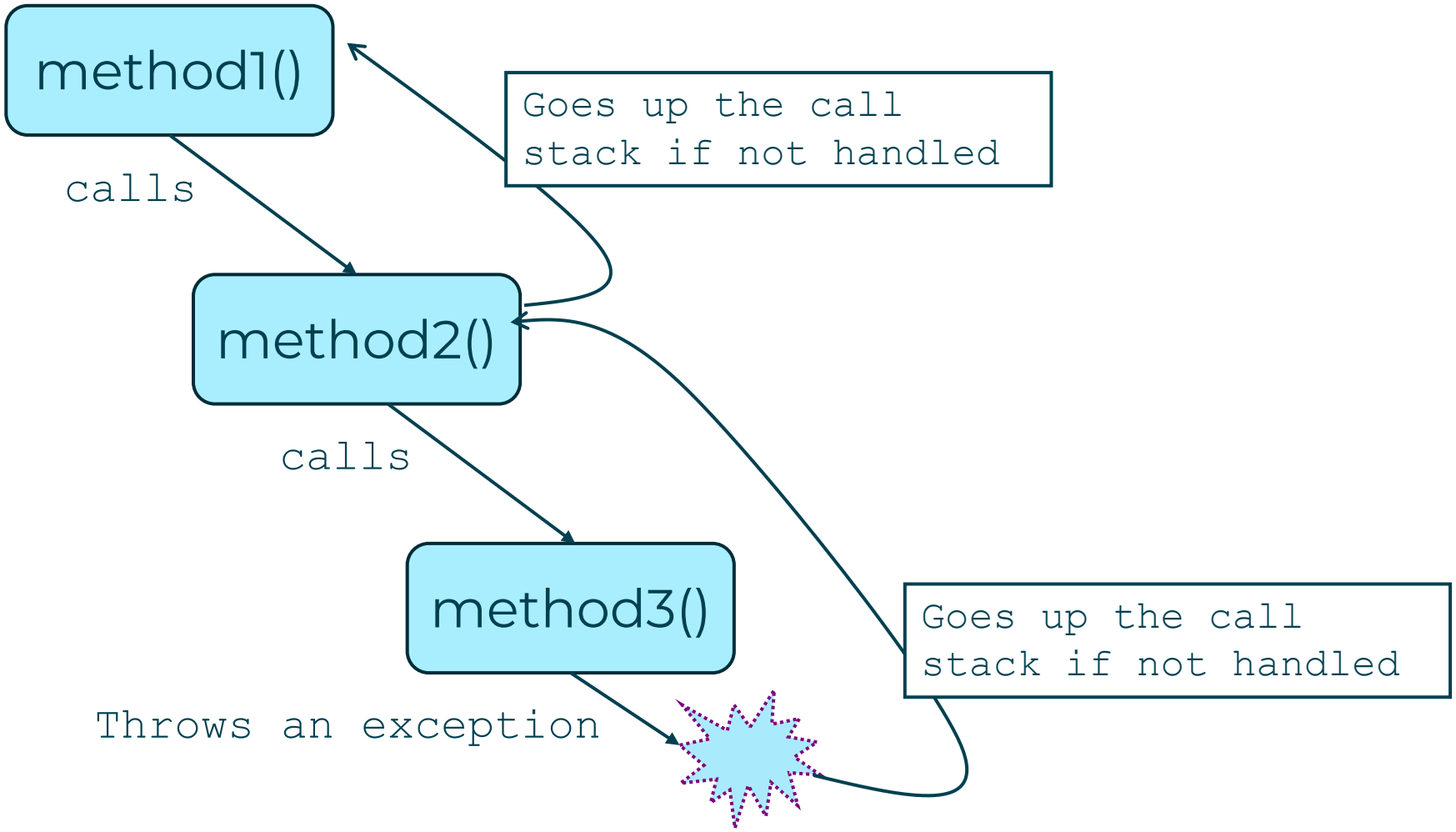
A few unpredictable exceptions

- **Accessing a file**
  - AccessDeniedException
  - FileNotFoundException
- **Accessing objects**
  - NullReferenceException
- **Networking**
  - SSLException
  - ConnectException
  - SocketTimeoutException

TYPES OF EXCEPTIONS AND ERRORS

- **class Throwable is the base class of all Exceptions and Errors. Three main types:**
  - Error
    - Typically unrecoverable external error
      - Out of memory, stack overflow etc.
    - Unchecked by compiler
  - **RuntimeException**
    - Typically programming error
    - Unchecked by compiler
  - **Exception**
    - Recoverable error (Database/File IO etc.)
    - *Checked* by Java compiler (must be caught or thrown)

Exceptions bubble up the call stack



Example of Exception bubbling up

Same coding error but exception thrown in called method

```
public static void main(String[] args) {
        int[] ages = new int[7];
        processAges(ages);
}

public static void processAges(int[] ages) {
        print("Last age is" + ages[7]);
}
```

exception

Console output

```
Exception in thread "main"
ArrayIndexOutOfBoundsException: 7
```

try/catch/finally syntax

```
try {
  // guarded block
  execute_code();
}
catch( SomeSpecificExceptionType exn) {
       print("....: " + exn.getMessage());

}
catch( Exception exn ) {  // catch everything else
       print("General error: " + exn.getMessage());

}
finally {
  // closing files/connections
}
//remainder of containing method
```

Clean up and/or abort

Execute this whatever happens

Multiple Exceptions example

Before Java-7

```
try {
        // some code
} catch (IOException ex) {
        logger.error(ex);
        throw new Exception("Cannot open file");
} catch (SQLException ex) {
        logger.error(ex);
        throw new Exception("Database error!");
}
```

Inform the caller

Java-7

```
try {
        // some code
} catch(IOException | SQLException ex){
        logger.error(ex);
        throw new Exception("Data access error!");
}
```

METHOD THROWING AN EXCEPTION .. ‘THROWS’

- **If a method:**
  - Contains a statement that throws a checked exception
  - Calls a method that throws an unhandled checked exception
- **Then it must ‘declare itself’ as throwing a checked exception**
  - Enables the compiler to see that a catch clause is needed
- **Let's see a code example…**

Method throwing checked exceptions

```
public static void main(String[] args) {
    try {
        readFile();
    } catch (FileNotFoundException e) {
        // code to handle exception
    } catch (IOException e) {
        // code to handle exception
    }
}
```

```
private static void readFile() throws FileNotFoundException, IOException {
    File file = new File("test.txt");
    BufferedReader br = new BufferedReader(new FileReader(file));

    String st;
    while ((st = br.readLine()) != null) {
        System.out.println(st);
    }
}
```

throws FileNotFoundException

throws IOException

readFile() should either catch or throw the exceptions

Java SE 7: Try-with-Resource

- **A new try-statement that releases resources on termination**

```
try (FileInputStream fis = new FileInputStream(file)) {
   ...
} //fis will be closed() here
catch (Exception ex) {
   ...
}
```

- Resource must implement the **`java.lang.AutoCloseable`** interface
  - **`close()`** method releases the resource

Best Practice Guidance

- **Don't only catch `class Throwable or Exception`**
- **Don't try to catch every possible exception type**
  - Have a ‘final’ catch of `Exception`
- **You don't need `try {} catch {}` in every method**
- **You will use `finally` blocks to ensure resources freed**
- **Only use exceptions for *exceptional* circumstances**
- **Be wary of just reporting back large error message**
  - Might contain system information
  - Might be better with “Sorry, we couldn’t find those credentials”

Review

- **Java uses exceptions to report errors**
  - Use `try` / `catch` / `finally` blocks to encapsulate such code
  - You can throw (or re-throw caught) exceptions
- **Unhandled exceptions will be caught by the Java runtime**

Hands-On Labs

- Working with exceptions

Understanding Execution Flow – 1

```
public class Program {
   static void main(String[] args) {
      try {
         Task.f1( 0 );
         Task.f2();
      }
      catch( Exception exn )
      {
         System.out.println(exn.getMessage() );
      }
   }
}
```

```
public class Task {
   public static void f1( int a ) {
      f3( a );
      f4();
   }

   public static void f2() { ... }

   public static void f3( int y ) {
      int x = 10 / y;
      ...
   }
   public static void f4() { ... }
}
```

Step

Understanding Execution Flow – 2

```
public class Program {
   static void main() {
      try {
         Task.f1( 0 );
         Task.f2();
      }
      catch( Exception exn )
      {
         Syetem.out.println
          exn.getMessage() );
      }
   }
}
```

```
public class Task {
   public static void f1( int a ) {
      f3( a );
      f4();
   }

   public static void f2() { ... }

   public static void f3( int y ) {
      int x;
      try {
         x = 10 / y;
         ... // Does not run
      }
      catch( ArithmeticException exn )
      {…}
      // Rest of method
   }
   public static void f4() { ... }
}
```

Step

Understanding Execution Flow – 3

```
public class Program {
   static void main() {
      try {
         Task.f1( 0 );
         Task.f2();
      }
      catch( Exception exn )
      {
         System.out.println
          (exn.getMessage());
      }
   }
}
```

```
public class Task {
   public static void f1( int a ) {
      f3( a );
      f4();
   }

   public static void f2() { ... }

   public static void f3( int y ) {
      int x;
      try {
        x = 10 / y;
        System.out.println( “AAA" );
      }
      finally {
        System.out.println( “BBB" );
      }
     // does not run if try fails
   }
   public static void f4() { ... }
}
```

Step

Throwing Exceptions

- **To 'raise' an exception, we throw (an instance of) it**
  - Pass information through constructor arguments

```
void printReport( Report rpt ) {
   if( rpt == null ) {
      throw new IllegalArgumentException(
       “’Report’ parameter null, can't print null report");
   }
   ...
}
```

Predefined Java Exception!

Don’t forget ‘new’

- **You can re-throw a caught exception**
  - This maintains original stack location where exception thrown

```
catch(IllegalArgumentException exn )
{
   ...
  throw exn;
}
```

Re-throw!

Finally clause – revisited

```
InputStream in = null;
try {
    System.out.println("We are opening a file");
    in = new FileInputStream("ThisFileIsMissing.txt");
    System.out.println("File open");
    int data = in.read();
} catch (IOException ioe) {
    System.out.println(ioe.getMessage());
} finally {
    try {
        if(in != null) in.close();
    } catch(IOException e) {
        System.out.println("Failed to close file");
    }
}
```

Runs regardless

Must close resources

Java: try-with-resources –another example

Java SE 7 introduced a new statement that ‘auto closes’ resources

- Can eliminate the need for a lengthy finally block

```
try (InputStream in = new FileInputStream(“ThisFileIsMissing.txt"))
{
      System.out.println("File open");
      int data = in.read();

} catch (FileNotFoundException fnfe) {
    System.out.println(fnfe.getMessage());

} catch (IOException ioe) {
    System.out.println(ioe.getMessage());
}
```

Suppressed Exceptions

If exception occurs in try block of try-with-resources

- An exception occurs while closing the resources
- The resulting (close) exception is suppressed

(So original exception of the 'try' is the one that a catch would handle)

```
} catch (Exception e) {
    System.out.println(e.getMessage());
    for(Throwable t : e.getSuppressed()) {
        System.out.println(t.getMessage());
    }
}
```