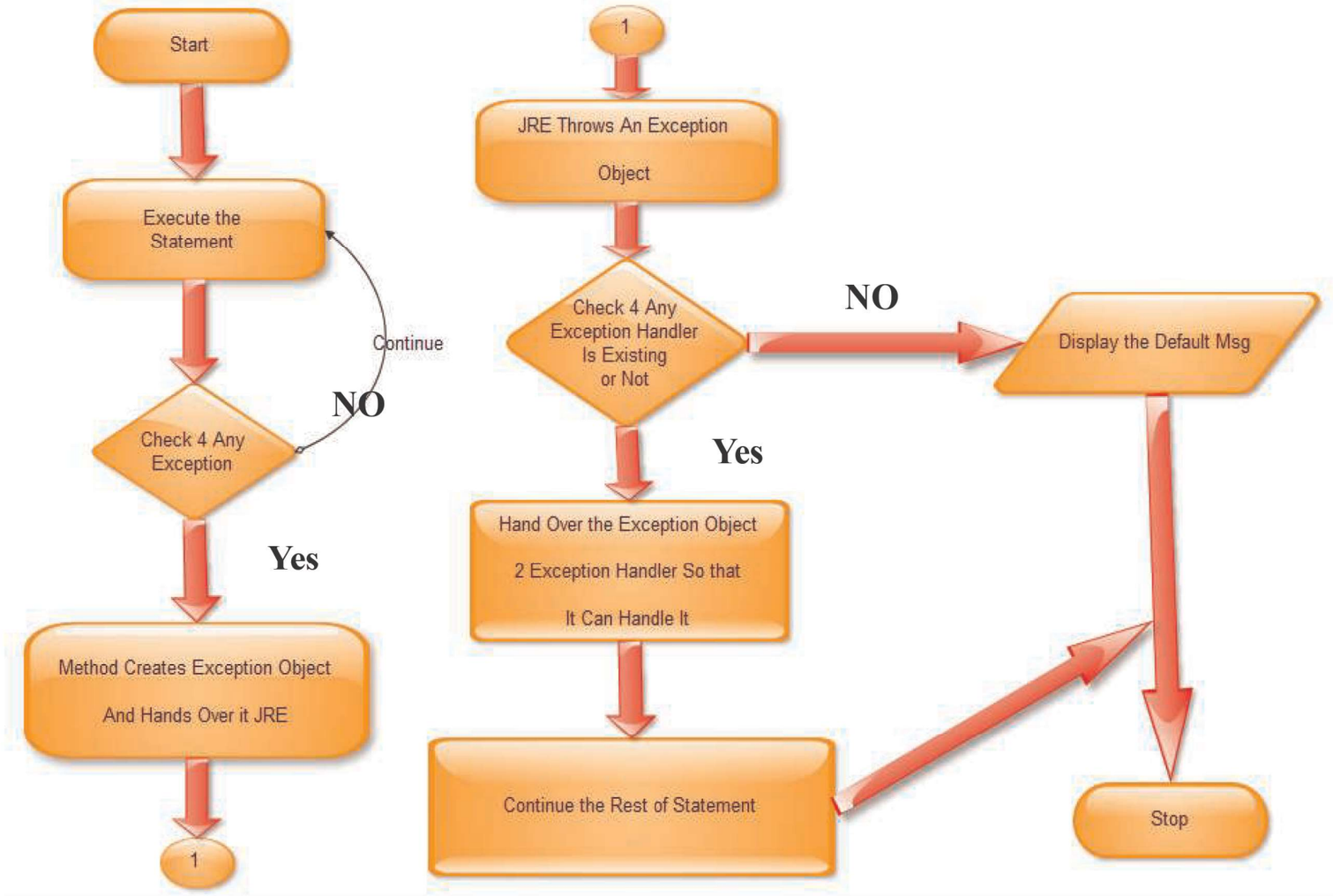
14

Throwing and Catching Exceptions

Objectives

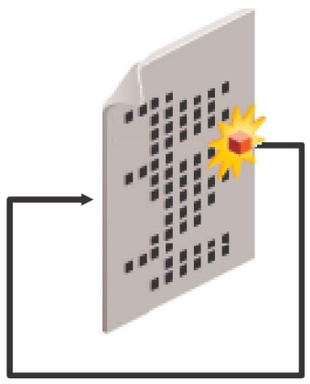
After completing this lesson, you should be able to do the following:

- Explain the basic concepts of exception handling
- Write code to catch and handle exceptions
- Write code to throw exceptions
- Create your own exceptions

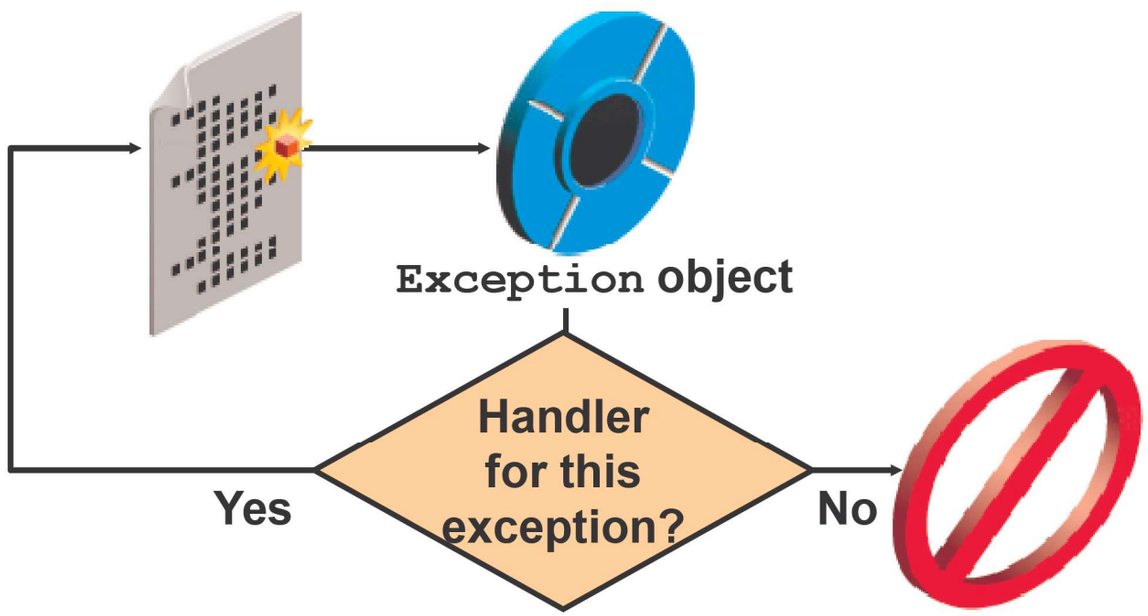

Start
Execute the Statement
Check 4 Any Exception
Continue
NO
Yes
Method Creates Exception Object
And Hands Over it JRE
1
1
JRE Throws An Exception Object
Check 4 Any Exception Handler Is Existing or Not
NO
Display the Default Msg
Yes
Hand Over the Exception Object
2 Exception Handler So that
It Can Handle It
Continue the Rest of Statement
Stop

What Is an Exception?

- An exception is an unexpected event.



1. A method throws an exception.
2. A handler catches the exception.

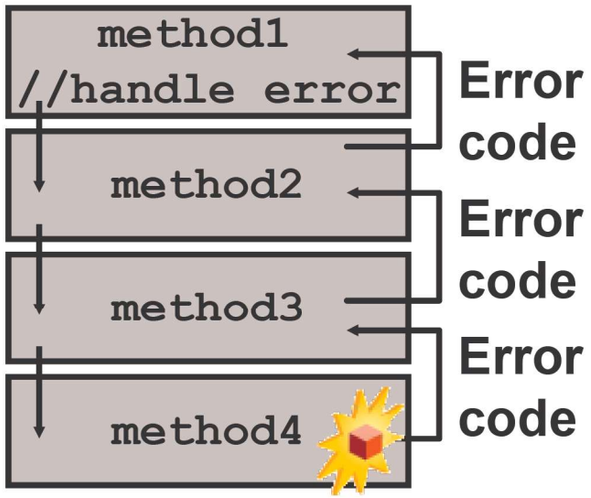


Advantages of Java Exceptions: Separating Error-Handling Code

- In traditional programming, error handling often makes code more confusing to read.
- Java separates the details of handling unexpected errors from the main work of the program.
- The resulting code is clearer to read and, as a result, less prone to bugs.

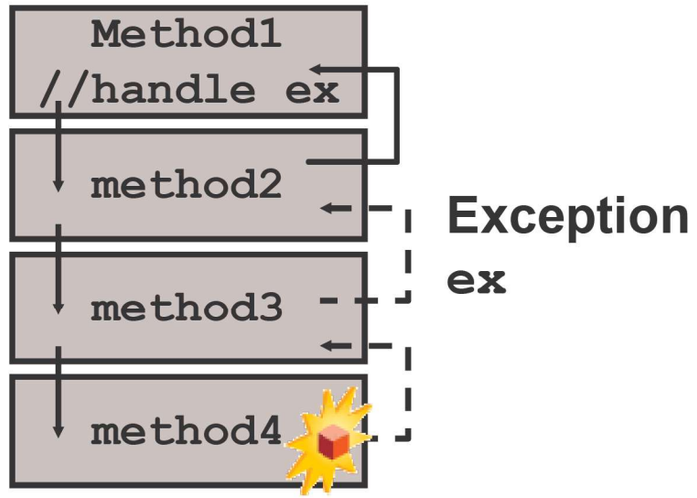
Advantages of Java Exceptions: Passing Errors Up the Call

Traditional error handling



Each method checks for errors and returns an error code to its calling method.

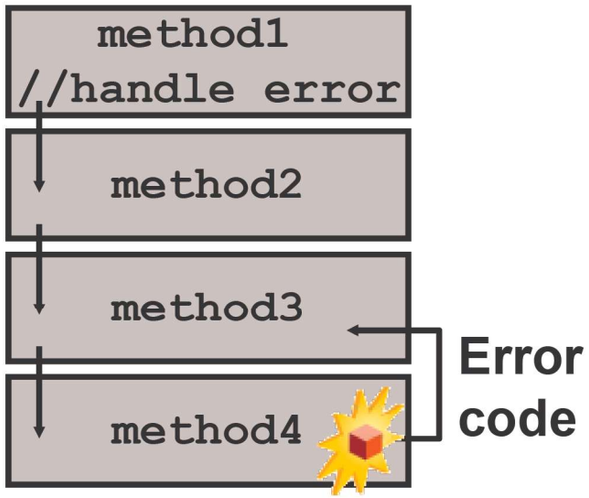
Java exceptions



`method4` throws an exception; eventually `method1` catches it.

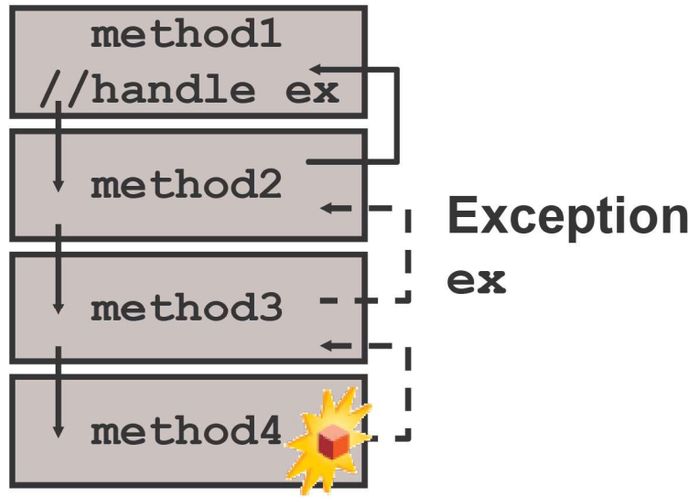
Advantages of Java Exceptions: Exceptions Cannot Be Ignored

Traditional error handling



If `method3` ignores the error, it will never be handled.

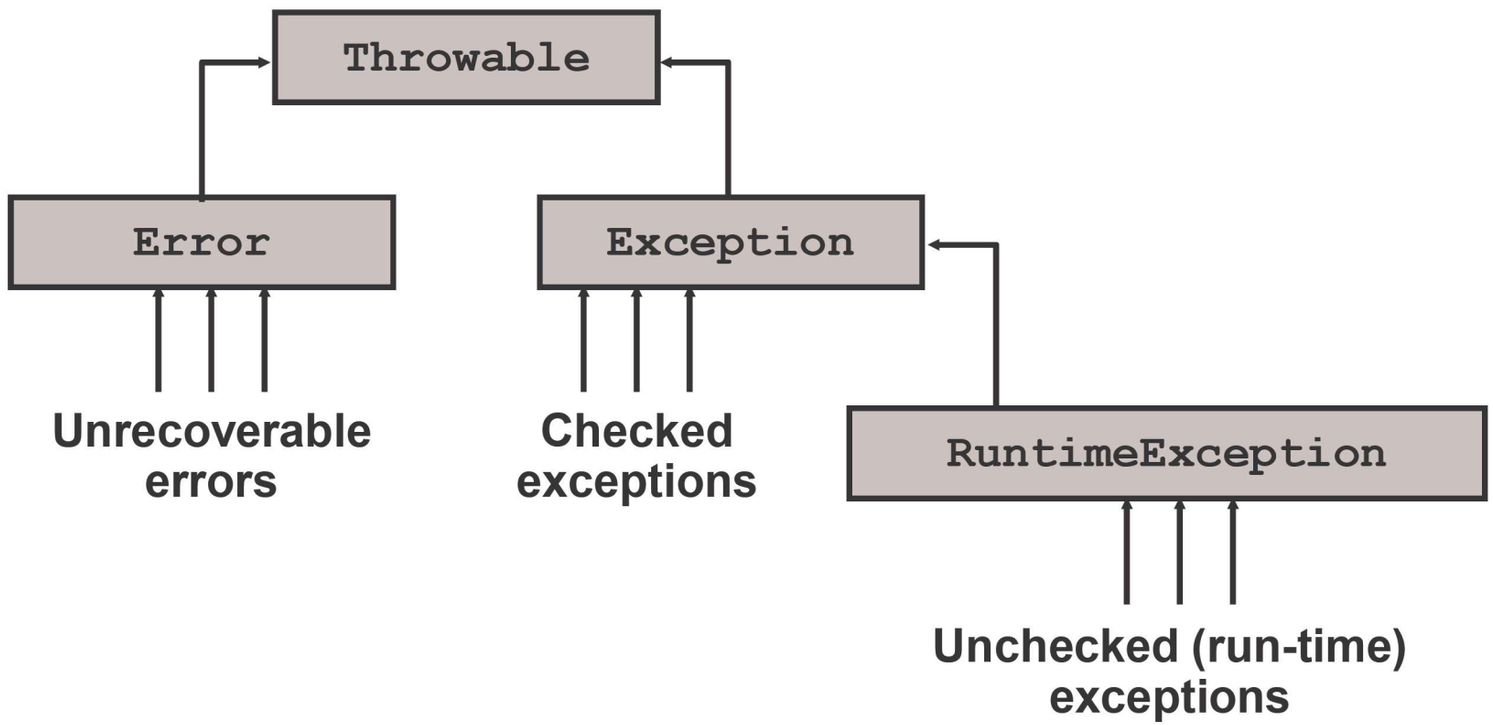
Java exceptions



The exception must be caught and handled somewhere.

Checked Exceptions, Unchecked Exceptions, and Errors

- All errors and exceptions extend the `Throwable` class.



Handling Exceptions

- Three choices:
  - Catch the exception and handle it.
  - Allow the exception to pass to the calling method.
  - Catch the exception and throw a different exception.

Catching and Handling Exceptions

- Enclose the method call in a `try` block.
- Handle each exception in a `catch` block.
- Perform any final processing in a `finally` block.

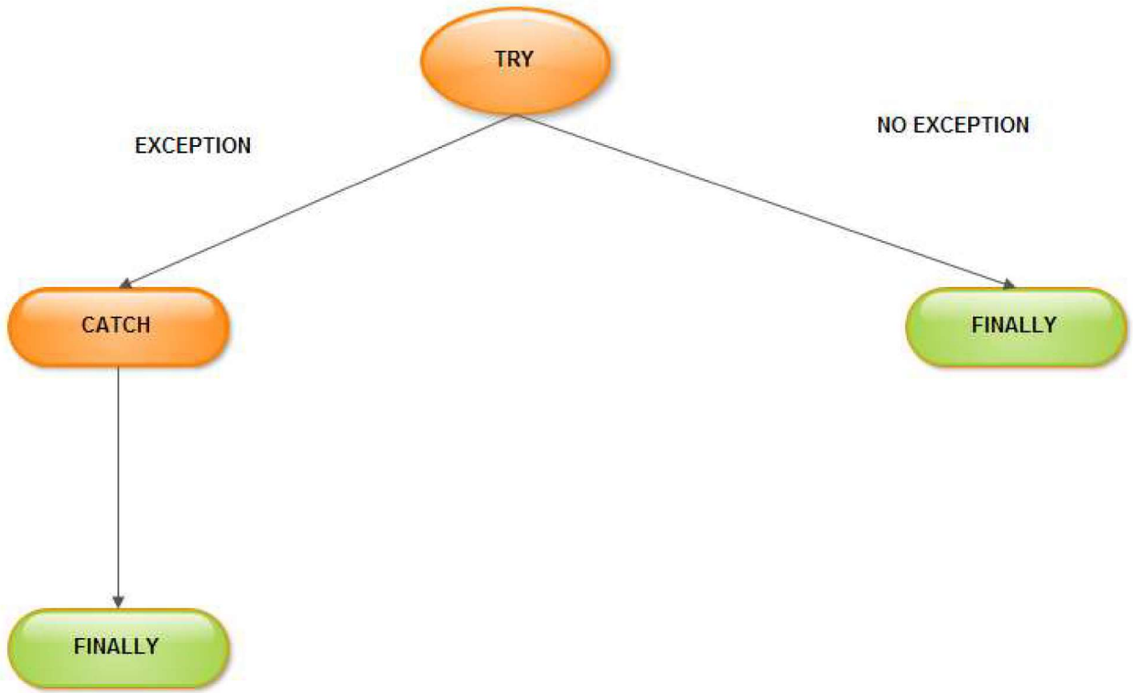
```
try {
  // call the method
}
catch (exception1) {
  // handle exception1
}
catch (exception2) {
  // handle exception2
}…
finally {
  // any final processing
}
```

Catching a Single Exception

```
int qty;
String s = getQtyFromForm();
try {
  // Might throw NumberFormatException
  qty = Integer.parseInt(s);
}
catch ( NumberFormatException e ) {
  // Handle the exception
}
// If no exceptions were thrown, we end up here
```

Catching Multiple Exceptions

```
try {
  // Might throw MalformedURLException
  URL u = new URL(str);
  // Might throw IOException
  URLConnection c = u.openConnection();
}
catch (MalformedURLException e) {
  System.err.println("Could not open URL: " + e);
}
catch (IOException e) {
  System.err.println("Could not connect: " + e);
}
```


TRY
EXCEPTION
NO EXCEPTION
CATCH
FINALLY
FINALLY

Cleaning Up with a `finally` Block

```
FileInputStream f = null;
try {
  f = new FileInputStream(filePath);
  while (f.read() != -1)
    charcount++;
}
catch(IOException e) {
  System.out.println("Error accessing file " + e);
}
finally {
  // This block is always executed
  f.close();
}
```

Guided Practice: Catching and

```
void makeConnection(String url) {
  try {
    URL u = new URL(url);
  }
  catch (MalformedURLException e) {
    System.out.println("Invalid URL: " + url);
    return;
  }
  finally {
    System.out.println("Finally block");
  }
  System.out.println("Exiting makeConnection");
}
```

Guided Practice: Catching and Handling Exceptions

```
void myMethod () {
  try {
    getSomething();
  } catch (IndexOutOfBoundsException e1) {
     System.out.println("Caught IOBException ");
  } catch (Exception e2) {
     System.out.println("Caught Exception ");
  } finally {
     System.out.println("No more exceptions ");
  }
  System.out.println("Goodbye");
}
```

The `try`-with-resources Statement

Java SE 7 provides a new `try`-with-resources statement that will autoclose resources.

```
System.out.println("About to open a file");
try (InputStream in =
      new FileInputStream("missingfile.txt")) {
    System.out.println("File open");
    int data = in.read();
} catch (FileNotFoundException e) {
    System.out.println(e.getMessage());
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

The `AutoCloseable` Interface

Resource in a `try`-with-resources statement must implement either:

- `java.lang.AutoCloseable`
  - New in JDK 7
  - May throw an `Exception`
- `java.io.Closeable`
  - Refactored in JDK7 to extend `AutoCloseable`
  - May throw an `IOException`

```
public interface AutoCloseable {
    void close() throws Exception;
}
```

Catching Multiple Exceptions

Java SE 7 provides a new multi-`catch` clause.

```
ShoppingCart cart = null;
try (InputStream is = new FileInputStream(cartFile);
     ObjectInputStream in = new ObjectInputStream(is)) {
     cart = (ShoppingCart)in.readObject();
} catch (ClassNotFoundException | IOException e) {
    System.out.println("Problem serializing " +
   cartFile);
    System.out.println(e);
    System.exit(-1);
}
```

Multiple exception types are separated with a vertical bar.

Allowing an Exception to Pass to the

- Use `throws` in the method declaration.
- The exception propagates to the calling method.

```
public int myMethod() throws exception1 {
  // code that might throw exception1
}
```

```
public URL changeURL(URL oldURL)
                    throws MalformedURLException {
  return new URL("http://www.oracle.com");
}
```

Throwing Exceptions

- Throw exceptions by using the `throw` keyword.
- Use `throws` in the method declaration.

```
throw new Exception1();
```

```
public String getValue(int index) throws
                      IndexOutOfBoundsException {
  if (index < 0 || index >= values.length) {
    throw new IndexOutOfBoundsException();
  }
  …
}
```

Creating Exceptions

- Extend the `Exception` class:

```
public class MyException extends Exception { … }
```

```
public class UserFileException extends Exception {
  public UserFileException (String message) {
     super(message);
  }
}
```

Catching an Exception and Throwing a Different Exception

```
catch (exception1 e) {
  throw new exception2(…);
}
```

```
void readUserFile() throws UserFileException {
  try {
    // code to open and read userfile
  }
  catch(IOException e) {
    throw new UserFileException(e.toString());
  }
}
```

Summary

In this lesson, you should have learned how to do the following:

- Use Java exceptions for robust error handling
- Handle exceptions by using `try`, `catch`, and `finally`
- Use the `throw` keyword to throw an exception
- Use a method to declare an exception in its signature to pass it up the call stack

Practice : Overview

This practice covers the following topics:

- Creating a custom exception
- Changing `DataMan` finder methods to throw exceptions
- Handling the exceptions when calling `DataMan` finder methods
- Testing the changes to the code