Handling errors

Madhavan Mukund, S P Suresh

Programming Language Concepts

Lecture 10, 13 February 2025

When things go wrong

- Our code could encounter many types of errors
  - *User input* — enter invalid filenames or URLs
  - *Device errors* — printer jam, network connection drops
  - *Resource limitations* — disk full
  - *Code errors* — invalid array index, key not present in hash table, refer to a variable that is `null`, divide by zero, . . .

When things go wrong

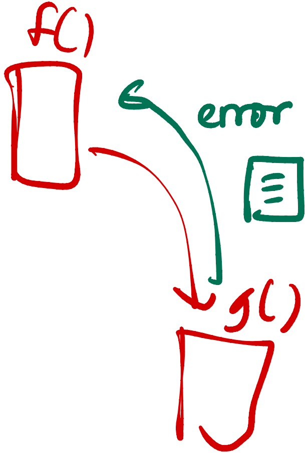
- Our code could encounter many types of errors
  - *User input* — enter invalid filenames or URLs
  - *Device errors* — printer jam, network connection drops
  - *Resource limitations* — disk full
  - *Code errors* — invalid array index, key not present in hash table, refer to a variable that is `null`, divide by zero, . . .
- Signalling errors
  - Return an invalid value: -1 at end of file, `null`
  - What if there is no obvious invalid value?

Exception handling

- Code that generates error raises or throws an exception

Exception handling

- Code that generates error raises or throws an exception
- Notify the type of error
  - Information about the nature of the exception
  - Natural to structure an exception as an object

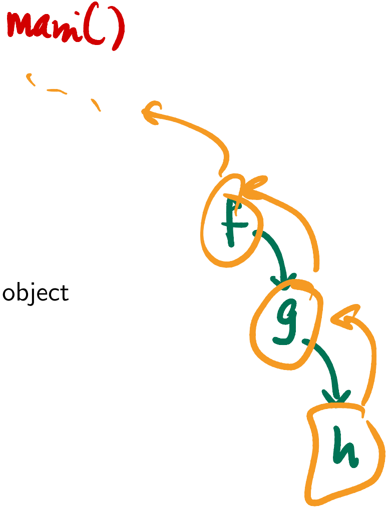


Exception handling

- Code that generates error raises or throws an exception
- Notify the type of error
  - Information about the nature of the exception
  - Natural to structure an exception as an object
- Caller catches the exception and takes corrective action
  - Extract information about the error from the exception object
  - Graceful interruption rather than program crash

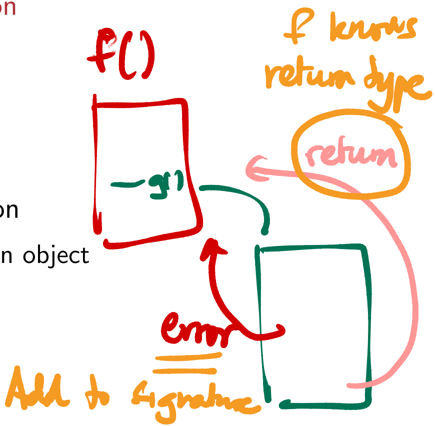
Exception handling

- Code that generates error raises or throws an exception
- Notify the type of error
  - Information about the nature of the exception
  - Natural to structure an exception as an object
- Caller catches the exception and takes corrective action
  - Extract information about the error from the exception object
  - Graceful interruption rather than program crash
- . . . or passes the exception back up the calling chain



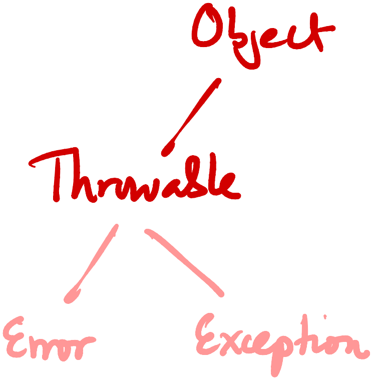
Exception handling

- Code that generates error raises or throws an exception
- Notify the type of error
  - Information about the nature of the exception
  - Natural to structure an exception as an object
- Caller catches the exception and takes corrective action
  - Extract information about the error from the exception object
  - Graceful interruption rather than program crash
- . . . or passes the exception back up the calling chain
- Declare if a method can throw an exception
  - Compiler can check whether calling code has made a provision to handle the exception



Java's classification of errors

- All exceptions descend from class `Throwable`
  - Two branches, `Error` and `Exception`



Java's classification of errors

- All exceptions descend from class `Throwable`
  - Two branches, `Error` and `Exception`
- `Error` — relatively rare, "not the programmer's fault"
  - Internal errors, resource limitations within Java runtime
  - No realistic corrective action possible, notify caller and terminate gracefully

Java's classification of errors

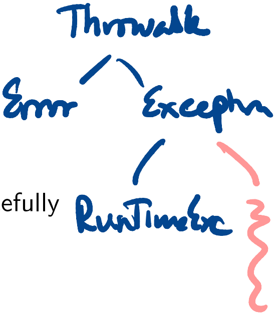
- All exceptions descend from class `Throwable`
  - Two branches, `Error` and `Exception`
- `Error` — relatively rare, "not the programmer's fault"
  - Internal errors, resource limitations within Java runtime
  - No realistic corrective action possible, notify caller and terminate gracefully
- `Exception` — two sub branches
  - `RunTimeException`, checked exceptions

Java's classification of errors

- All exceptions descend from class `Throwable`
  - Two branches, `Error` and `Exception`
- `Error` — relatively rare, "not the programmer's fault"
  - Internal errors, resource limitations within Java runtime
  - No realistic corrective action possible, notify caller and terminate gracefully
- `Exception` — two sub branches
  - `RunTimeException`, checked exceptions
- `RunTimeException` — programming errors that should have been caught by code
  - Array index out of bounds, invalid hash key, . . .

Java's classification of errors

- All exceptions descend from class `Throwable`
  - Two branches, `Error` and `Exception`
- `Error` — relatively rare, "not the programmer's fault"
  - Internal errors, resource limitations within Java runtime
  - No realistic corrective action possible, notify caller and terminate gracefully
- `Exception` — two sub branches
  - `RunTimeException`, checked exceptions
- `RunTimeException` — programming errors that should have been caught by code
  - Array index out of bounds, invalid hash key, ...
- Checked exceptions
  - Typically user-defined, code assumptions violated
    - In a list of orders, quantities should be positive integers



Catching and handling exceptions

- try–catch
  - Enclose code that may generate exception in a try block
  - Exception handler in catch block
  - Similar to Python

```
try {
    ...
    call a function that may
      throw an exception
    ...
}
catch (ExceptionType e){
    ...
    examine e and handle it
    ...
}
```

Catching and handling exceptions

- try–catch
  - Enclose code that may generate exception in a try block
  - Exception handler in catch block
  - Similar to Python
- If try encounters an exception, rest of the code in the block is skipped

```
try {
  ...
  call a function that may
    throw an exception
  ..
}
catch (ExceptionType e){
  ...
  examine e and handle it
  ...
}
```

s1
g()
s2
error

Catching and handling exceptions

- `try`–`catch`
  - Enclose code that may generate exception in a `try` block
  - Exception handler in `catch` block
  - Similar to Python
- If `try` encounters an exception, rest of the code in the block is skipped
- If exception matches the type in `catch`, handler code executes

```
try {
  ...
  call a function that may
    throw an exception
  ..
}
catch (ExceptionType e){
  ...
  examine e and handle it
  ...
}
```

Catching and handling exceptions

- try–catch
  - Enclose code that may generate exception in a try block
  - Exception handler in catch block
  - Similar to Python
- If try encounters an exception, rest of the code in the block is skipped
- If exception matches the type in catch, handler code executes
- Otherwise, uncaught exception is passed back to the code that called this code

```
try {
  ...
  call a function that may
    throw an exception
  ..
}
catch (ExceptionType e){
  ...
  examine e and handle it
  ...
}
```

Catching and handling exceptions

- try–catch
  - Enclose code that may generate exception in a try block
  - Exception handler in catch block
  - Similar to Python
- If try encounters an exception, rest of the code in the block is skipped
- If exception matches the type in catch, handler code executes
- Otherwise, uncaught exception is passed back to the code that called this code
- Top level uncaught exception — program crash

```
try {
  ...
  call a function that may
    throw an exception
  ..
}
catch (ExceptionType e){
  ...
  examine e and handle it
  ...
}
```

Catching and handling exceptions

- Can catch more than one type of exception
  - Multiple `catch` blocks

```
try {
  code that might throw exceptions
}
catch (FileNotFoundException e) {
  handle missing files
}
catch (UnknownHostException e) {
  handle unknown hosts
}
catch (IOException e) {
  handle all other I/O issues
}
```

Catching and handling exceptions

- Can catch more than one type of exception
    - Multiple `catch` blocks
- Exceptions are classes in the Java class hiearachy
    - `catch (ExceptionType e)` matches any subtype of `ExceptionType`

```
try {
  code that might throw exceptions
}
catch (FileNotFoundException e) {
  handle missing files
}
catch (UnknownHostException e) {
  handle unknown hosts
}
catch (IOException e) {
  handle all other I/O issues
}
```

o()

abort

Catching and handling exceptions

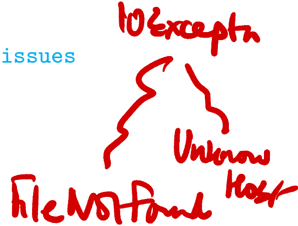
- Can catch more than one type of exception
  - Multiple `catch` blocks
- Exceptions are classes in the Java class hiearachy
  - `catch (ExceptionType e)` matches any subtype of `ExceptionType`
- Catch blocks are tried in sequence
  - Match exception type against each one in turn

```
try {
  code that might throw exceptions
}
catch (FileNotFoundException e) {
  handle missing files
}
catch (UnknownHostException e) {
  handle unknown hosts
}
catch (IOException e) {
  handle all other I/O issues
}
```

Catching and handling exceptions

- Can catch more than one type of exception
  - Multiple `catch` blocks
- Exceptions are classes in the Java class hiearachy
  - `catch (ExceptionType e)` matches any subtype of `ExceptionType`
- Catch blocks are tried in sequence
  - Match exception type against each one in turn
- Order `catch` blocks by argument type, more specific to less specific
  - `IOException` would intercept `FileNotFoundException`

```
try {
  code that might throw exceptions
}
catch (FileNotFoundException e) {
  handle missing files
}
catch (UnknownHostException e) {
  handle unknown hosts
}
catch (IOException e) {
  handle all other I/O issues
}
```



Generating exceptions

- When does a function generate an exception?

Generating exceptions

- When does a function generate an exception?
- `Error` — JVM runtime issue

Generating exceptions

- When does a function generate an exception?
- `Error` — JVM runtime issue
- `RunTimeException`
  - Array index out of bounds, invalid hash key, . . .

Generating exceptions

- When does a function generate an exception?
- `Error` — JVM runtime issue
- `RunTimeException`
    - Array index out of bounds, invalid hash key, . . .
- Code calls another function that generates an exception

Generating exceptions

- When does a function generate an exception?
- `Error` — JVM runtime issue
- `RunTimeException`
  - Array index out of bounds, invalid hash key, . . .
- Code calls another function that generates an exception
- Your code detects an error and generates an exception
  - `throw` a checked exception

a[i]

retrieve (a, i)

Notifying checked exceptions

- Example: you write a method `readData()`
  - Header line provides length of data
    - `Content-Length: 2048`
  - Actual data read is less than promised length

Notifying checked exceptions

- Example: you write a method `readData()`
  - Header line provides length of data
    - `Content-Length: 2048`
  - Actual data read is less than promised length
- Search Java documentation for suitable pre-defined exception
  - `EOFException`, subtype of `IOException`
  - “Signals that EOF has been reached unexpectedly during input”

Notifying checked exceptions

- Example: you write a method `readData()`
  - Header line provides length of data
    - `Content-Length: 2048`
  - Actual data read is less than promised length
- Search Java documentation for suitable pre-defined exception
  - `EOFException`, subtype of `IOException`
  - “Signals that EOF has been reached unexpectedly during input”
- Create an object of exception type and `throw` it

```
throw new EOFException();
```

```
EOFExceptn O;
O = new EOF Exceptn();
throw O;
```

Notifying checked exceptions

- Example: you write a method `readData()`
  - Header line provides length of data
    - `Content-Length: 2048`
  - Actual data read is less than promised length
- Search Java documentation for suitable pre-defined exception
  - `EOFException`, subtype of `IOException`
  - “Signals that EOF has been reached unexpectedly during input”
- Create an object of exception type and `throw` it

```
throw new EOFException();
```

- Can also pass a diagnostic message when constructing exception object

```
String errormsg = "Content-Length:" + contentlen + ", Received: " + rcvdlen;
throw new EOFException(errormsg);
```

Throwing exceptions . . .

- How does caller know that `readData()` generates `EOFException`?

Throwing exceptions . . .

- How does caller know that `readData()` generates `EOFException`?
- Declare exceptions thrown in header

```
String readData(Scanner in)
    throws EOFException {
  ...
  while (...) {
    if (!in.hasNext()) {
      // EOF encountered
      if (n < len) {
        String errmsg = ...
        throw new EOFException(errmsg);
      }
    ...
  }
  return(s);
}
```

Throwing exceptions . . .

- How does caller know that `readData()` generates `EOFException`?
- Declare exceptions thrown in header
- Can throw multiple types of exceptions

```
String readFile(String filename)
    throws FileNotFoundException,
           EOFException { ... }
```

```
String readData(Scanner in)
    throws EOFException {
  ...
  while (...) {
    if (!in.hasNext()) {
      // EOF encountered
      if (n < len) {
        String errmsg = ...
        throw new EOFException(errmsg);
      }
    ...
  }
  return(s);
}
```

Throwing exceptions . . .

- How does caller know that `readData()` generates `EOFException`?
- Declare exceptions thrown in header
- Can throw multiple types of exceptions

```
String readFile(String filename)
    throws FileNotFoundException,
           EOFException { ... }
```

- Can throw any subtype of declared exception type

```
String readFile(String filename)
    throws IOException { ... }
```

  - Can throw `FileNotFoundException`, `EOFException`, both subclasses of `IOException`

```
String readData(Scanner in)
    throws EOFException {
  ...
  while (...) {
    if (!in.hasNext()) {
      // EOF encountered
      if (n < len) {
        String errmsg = ...
        throw new EOFException(errmsg);
      }
    ...
  }
  return(s);
}
```

Throwing exceptions . . .

- Method declares the exceptions it throws

```
String readData(Scanner in)
    throws EOFException {
  ...
  while (...) {
    if (!in.hasNext()) {
      // EOF encountered
      if (n < len) {
        String errmsg = ...
        throw new EOFException(errmsg);
      }
    ...
  }
  return(s);
}
```

Throwing exceptions . . .

- Method declares the exceptions it throws
- If you call such a method, you must handle it

```
String readData(Scanner in)
    throws EOFException {
  ...
  while (...) {
    if (!in.hasNext()) {
      // EOF encountered
      if (n < len) {
        String errmsg = ...
        throw new EOFException(errmsg);
      }
    ...
  }
  return(s);
}
```

Throwing exceptions . . .

- Method declares the exceptions it throws
- If you call such a method, you must handle it
- ... or pass it on; your method should advertise that it throws the same exception

```
String readData(Scanner in)
    throws EOFException {
  ...
  while (...) {
    if (!in.hasNext()) {
      // EOF encountered
      if (n < len) {
        String errmsg = ...
        throw new EOFException(errmsg);
      }
    ...
  }
  return(s);
}
```

Throwing exceptions . . .

- Method declares the exceptions it throws
- If you call such a method, you must handle it
- ... or pass it on; your method should advertise that it throws the same exception
- Need not advertise unchecked exceptions
  - Error, RunTimeException

```
String readData(Scanner in)
    throws EOFException {
  ...
  while (...) {
    if (!in.hasNext()) {
      // EOF encountered
      if (n < len) {
        String errmsg = ...
        throw new EOFException(errmsg);
      }
    ...
  }
  return(s);
}
```

Throwing exceptions . . .

- Method declares the exceptions it throws
- If you call such a method, you must handle it
- ... or pass it on; your method should advertise that it throws the same exception
- Need not advertise unchecked exceptions
  - `Error`, `RunTimeException`
- Should not normally generate `RunTimeException`
  - Fix the error or report suitable checked exception

```
String readData(Scanner in)
    throws EOFException {
  ...
  while (...) {
    if (!in.hasNext()) {
      // EOF encountered
      if (n < len) {
        String errmsg = ...
        throw new EOFException(errmsg);
      }
    ...
  }
  return(s);
}
```

Customized exceptions

- Don’t want negative numbers in a `LinearList`

Customized exceptions

- Don't want negative numbers in a LinearList
- Define a new class extending Exception

```
public class NegativeException extends Exception{

  private int error_value;
   // Negative value that generated exception

  public NegativeException(String message, int i){
    super(message);  // Appeal to superclass
    error_value = i; // constructor to set message
  }

  public int report_error_value(){
    return error_value;
  }
}
```

e.report_error_value()

Neg Exc

catch(NegExc e)

Customized exceptions

- Don't want negative numbers in a LinearList
- Define a new class extending Exception
- Throw this from LinearList
  - Note that add advertises the fact that it throws a NegativeException

```
public class NegativeException extends Exception{
   ...
}

public class LinearList{
  ...
  public add(int i) throws NegativeException{
    ...
    if (i < 0){
      throw new NegativeException("Negative input",i);
    }
    ...
  }
}
```

More on catching exceptions

- Can extract information about the exception

```
try {
  ...
  call a function that may
    throw an exception
  ..
}
catch (ExceptionType e){
  ...
  String errormsg = e.getMessage();
  ...
}
```

More on catching exceptions

- Can extract information about the exception
- Chaining exceptions
  - Process and throw a new exception from `catch`

```
try {
  ...
  access database
  ..
}
catch (SQLException e){
  ...
  String errormsg =
     "database error" + e.getMessage();
  throw new ServletException(errormsg);
  ...
}
```

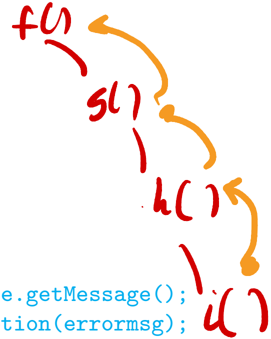
SQL Err

ServletErr

More on catching exceptions

- Can extract information about the exception
- Chaining exceptions
  - Process and throw a new exception from `catch`
- `Throwable` has additional methods to track chain of exceptions
  - `getCause()`, `initCause()`

```
try {
  ...
  access database
  ..
}
catch (SQLException e){
  ...
  String errormsg =
     "database error" + e.getMessage();
  throw new ServletException(errormsg);
  ...
}
```



More on catching exceptions

- Can extract information about the exception
- Chaining exceptions
  - Process and throw a new exception from `catch`
- `Throwable` has additional methods to track chain of exceptions
  - `getCause()`, `initCause()`
- Add information when you chain exceptions

```
try {
  ...
  access database
  ..
}
catch (SQLException e){
  ...
  String errormsg =
     "database error" + e.getMessage();
  ServletException newe =
     new ServletException(errormsg);
  newe.initCause(e);
  throw newe;
  ...
}
```

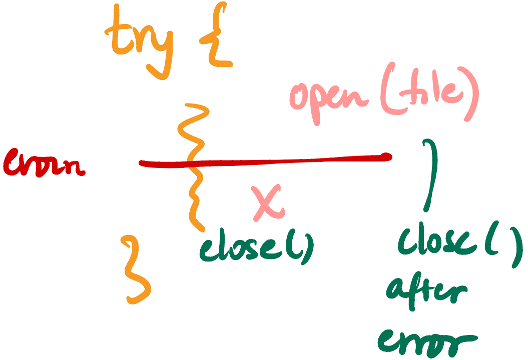
More on catching exceptions

- Can extract information about the exception
- Chaining exceptions
  - Process and throw a new exception from `catch`
- `Throwable` has additional methods to track chain of exceptions
  - `getCause()`, `initCause()`
- Add information when you chain exceptions
- Retrieve information when you catch exception

```
try {
  ...
}
catch (ServletException e){
  ...
  Throwable original = e.getCause();
  ...
}
```

Cleaning up resources

- When exception occurs, rest of the `try` block is skipped



Cleaning up resources

- When exception occurs, rest of the `try` block is skipped
- May need to do some clean up (close files, deallocate resources, . . .)

Cleaning up resources

- When exception occurs, rest of the try block is skipped
- May need to do some clean up (close files, deallocate resources, . . .)
- Add a block labelled finally

```
try{
     ...
   }

catch (ExceptionType1 e){...}

catch (ExceptionType2 e){...}

finally{
     ...
  // Always executed, whether try
  // terminates normally or
  // exceptionally. Use for clean up.
}
```

Cleaning up resources

- When exception occurs, rest of the try block is skipped
- May need to do some clean up (close files, deallocate resources, . . .)
- Add a block labelled finally
- Different scenarios

```
FileInputStream in =
  new FileInputStream(...);
try {
  // 1
  code that might throw exceptions
  // 2
}
catch (IOException e) {
  // 3
  show error message
  // 4
}
finally {
  // 5
  in.close();
}
// 6
```

Cleaning up resources

- When exception occurs, rest of the `try` block is skipped
- May need to do some clean up (close files, deallocate resources, . . .)
- Add a block labelled `finally`
- Different scenarios
  - No error — 1,2,5,6

```
FileInputStream in =
  new FileInputStream(...);
try {
  // 1
  code that might throw exceptions
  // 2
}
catch (IOException e) {
  // 3
  show error message
  // 4
}
finally {
  // 5
  in.close();
}
// 6
```

Cleaning up resources

- When exception occurs, rest of the `try` block is skipped
- May need to do some clean up (close files, deallocate resources, . . .)
- Add a block labelled `finally`
- Different scenarios
  - No error — 1,2,5,6
  - `IOException` in `try`, no exception in `catch` — 1,3,4,5,6

```
FileInputStream in =
  new FileInputStream(...);
try {
  // 1
  code that might throw exceptions
  // 2
}
catch (IOException e) {
  // 3
  show error message
  // 4
}
finally {
  // 5
  in.close();
}
// 6
```

Cleaning up resources

- When exception occurs, rest of the try block is skipped
- May need to do some clean up (close files, deallocate resources, . . .)
- Add a block labelled finally
- Different scenarios
  - No error — 1,2,5,6
  - IOException in try, no exception in catch — 1,3,4,5,6
  - IOException in try, chained exception in catch — 1,3,5

```
FileInputStream in =
  new FileInputStream(...);
try {
  // 1
  code that might throw exceptions
  // 2
}
catch (IOException e) {
  // 3
  show error message
  // 4
}
finally {
  // 5
  in.close();
}
// 6
```

& throw chained error