Programming in Java

Objectives

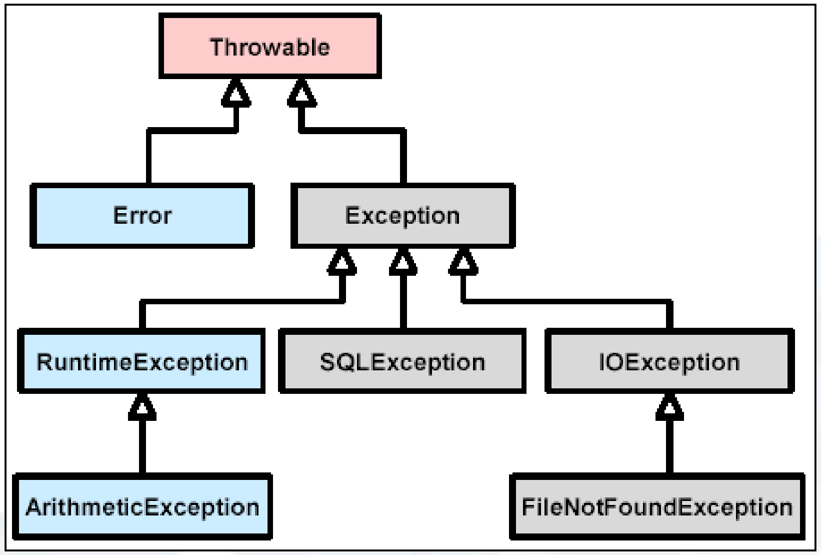
- In this session, you will learn to:
  - Explore errors and exceptions
  - Recognize exception classes and categories
  - Handle exceptions
  - Use the try-with-resources statement
  - Work with the AutoCloseable interface and supressed exceptions
  - Use the multi-catch clause
  - Use the throw clause
  - Create custom exceptions and use Wrapper exceptions
  - Use assertions and invariants

Errors and Exceptions

- Errors:
  - Handled to create reliable applications
  - Result of application bugs
  - Beyond the control of the application
- Exceptions:
  - An error that occurs at runtime
  - Abnormal behavior that leads to unpredictable result
  - Disrupts the normal flow of application execution

Exception Categories

- `java.lang.Throwable` class:
  - Parent class of all exceptions, as shown in the following figure.



  - Outlines several useful methods.
- Main categories of exceptions:
  - Unchecked exceptions
  - Checked exceptions

Exception Categories (Contd.)

- Unchecked exceptions:
  - Types:
    - `java.lang.RuntimeException`
    - `java.lang.Error`
  - Occur during the execution of application
  - Discovered by using the `try-catch` statement
  - Some `RuntimeExceptions`:
    - `ArrayIndexOutOfBoundsException`
    - `NullPointerException`
    - `ArithmeticException`
- Checked exceptions:
  - Type:
    - The `Exception`, except the `RuntimeException`
  - Handled using the `try` or the `throws` statement

Exception Handling

- Libraries that require knowledge of exception handling include:
  - File IO (NIO: `java.nio`)
  - Database access (JDBC: `java.sql`)
- `try-catch` blocks:
  - Used for handling exceptions
  - `try` block:
    - Handles exceptions
    - Sends the execution to the attached `catch` block
  - `catch` block:
    - Used to retry the operation
    - Used to try an alternate operation
    - Used to gracefully exit or return
    - Must not be empty
    - Gets a reference to the `java.lang.Exception` object

The try-catch Statement

- The following embedded Word document shows an example of using a `try-catch` statement.

try-catch statement

The try-catch Statement (Contd.)

- General purpose `catch` block:
  - Cannot deal with every possible error
  - Should not catch the base type of `Exception`
- Multiple `catch` blocks:
  - Can be associated with a single `try` block.
  - The following embedded Word document shows how multiple `catch` blocks can be associated with a single `try` block.

Multiple catch blocks

The finally Clause

- `finally` block:
  - Closes the opened resources
  - Executed with or without an error in the `try` block
  - Always executes after the `catch` block
  - May generate an `Exception`

The try-with-resources Statement

- `try`-with-resources statement:
  - Eliminates the need for a lengthy `finally` block
  - Always closes the opened resources
  - Allows opening of multiple resources
  - Closes multiple resources in the opposite order of opening
- A class that implements the `AutoCloseable` interface can be used as a resource.
- If a resource must be autoclosed, its reference must be declared within the `try` statement's parentheses.

The try-with-resources Statement (Contd.)

- The following embedded Word document shows how to declare a `try`-with-resources statement.

try-with-resources

The AutoCloseable Interface

- Resource in a `try`-with-resources statement must implement:
  - `java.lang.AutoCloseable`

    Or
  - `java.io.Closeable`
- The following code snippet shows how to declare the `AutoCloseable` interface with the `close()` method:

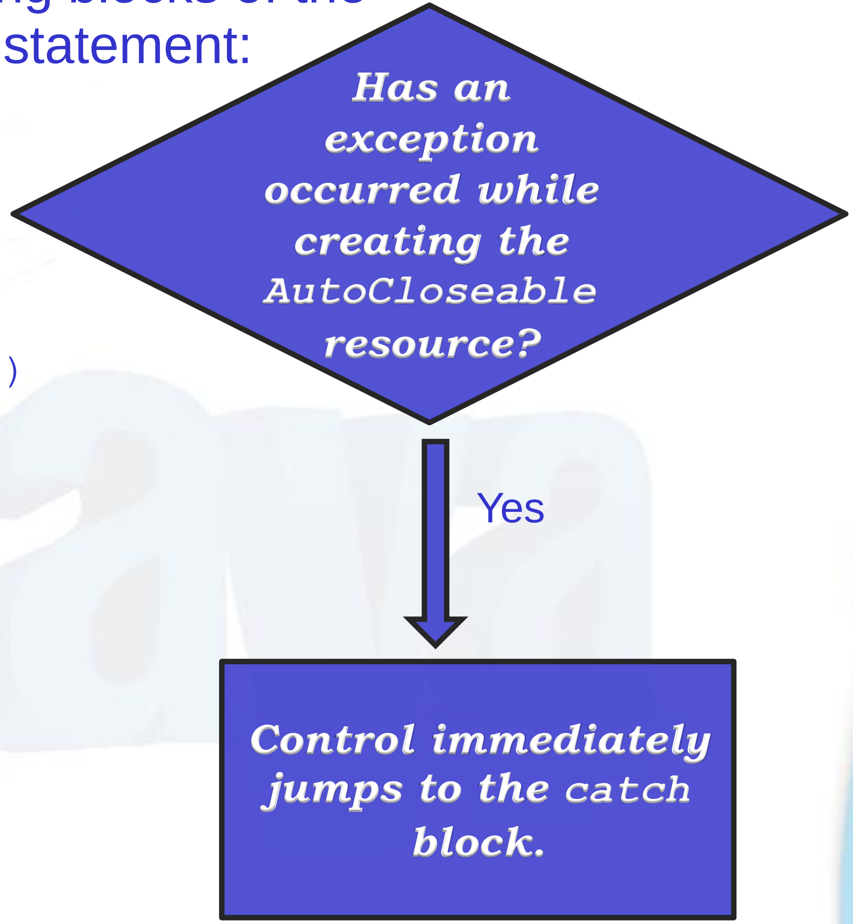
```
public interface AutoCloseable {
void close() throws Exception;
}
```

Programming in Java

Suppressed Exceptions

- Consider the following blocks of the `try`-with-resources statement:

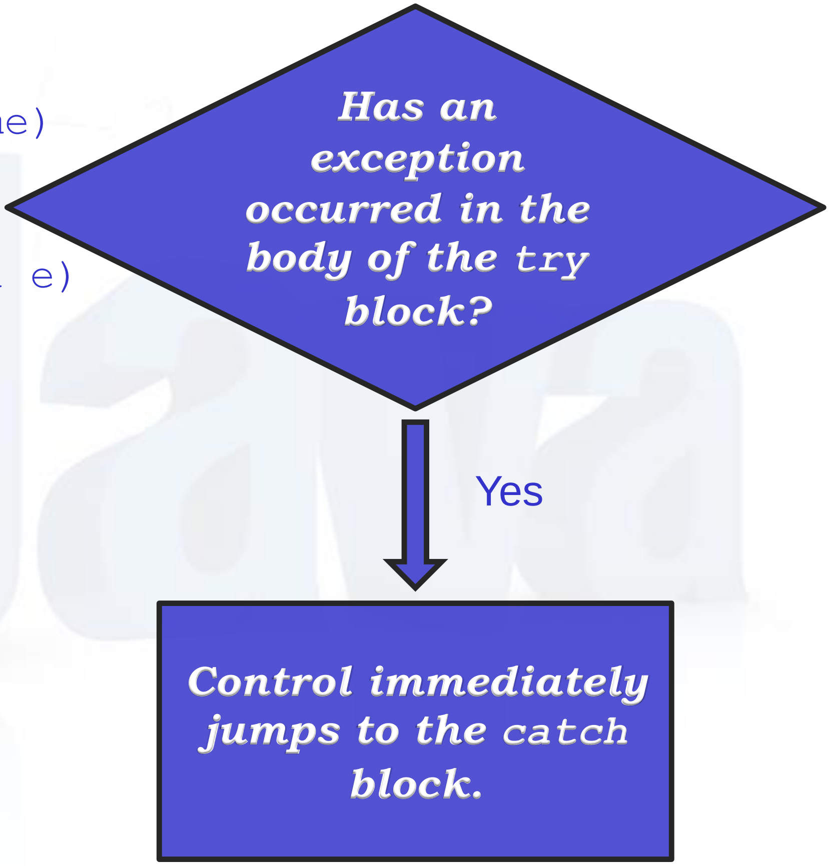
```
try(resource_name)
{
//Statements
}catch(Exception e)
{
//Statements
}
```


Has an exception occurred while creating the AutoCloseable resource?
Yes
Control immediately jumps to the catch block.

Suppressed Exceptions (Contd.)

- Consider the following syntax of the `try`-with-resources statement:

```
try(resource_name)
{
//Statements
}catch(Exception e)
{
//Statements
}
```

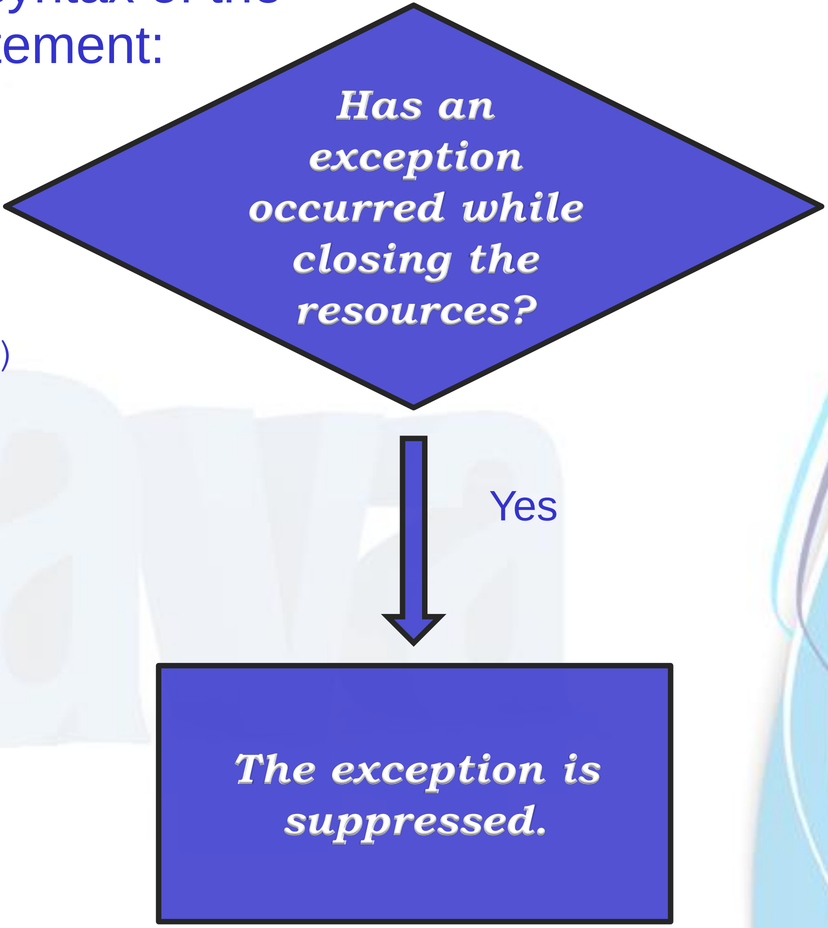

Has an exception occurred in the body of the try block?
Yes
Control immediately jumps to the catch block.

Programming in Java

Suppressed Exceptions (Contd.)

- Consider the following syntax of the `try`-with-resources statement:

```
try(resource_name)
{
//Statements
}catch(Exception e)
{
//Statements
}
```

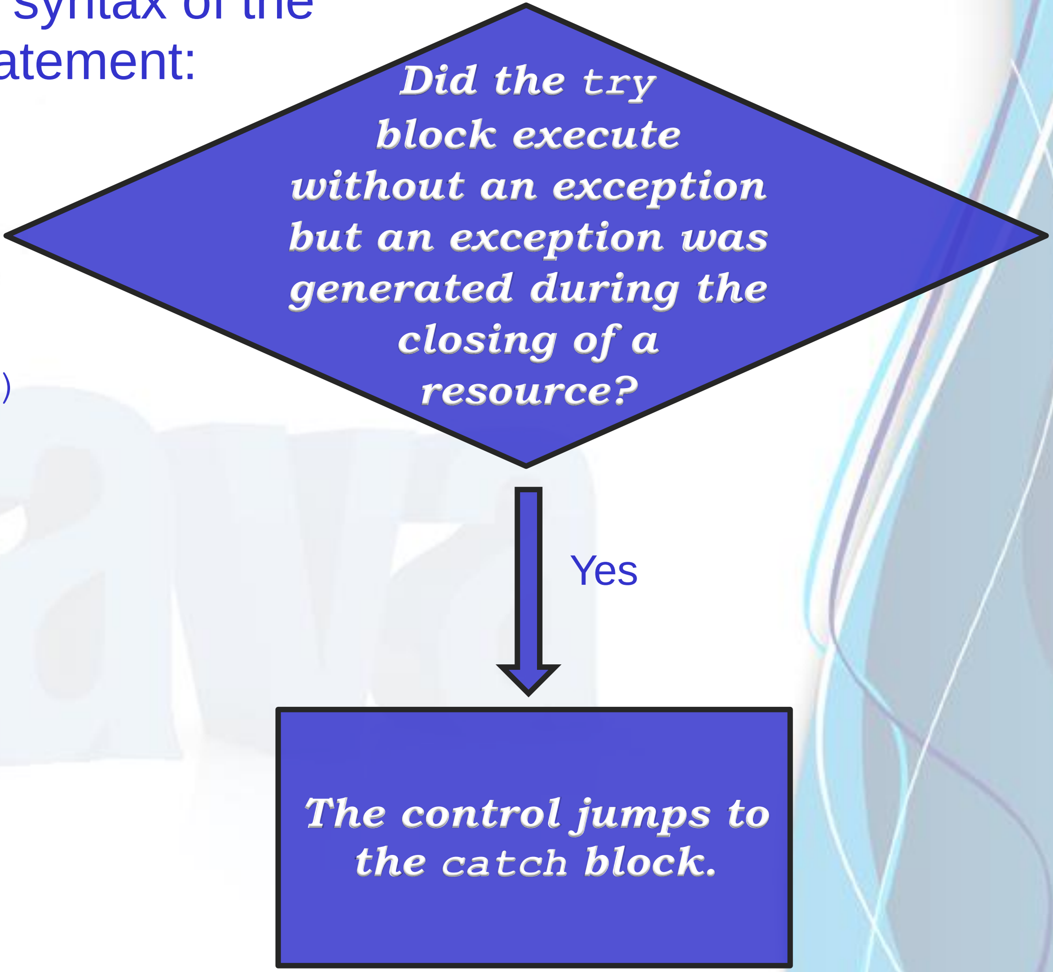


Has an exception occurred while closing the resources?
Yes
The exception is suppressed.

Suppressed Exceptions (Contd.)

- Consider the following syntax of the `try`-with-resources statement:

```
try(resource_name)
{
//Statements
}catch(Exception e)
{
//Statements
}
```


Did the try block execute without an exception but an exception was generated during the closing of a resource?
Yes
The control jumps to the catch block.

Suppressed Exceptions (Contd.)

- The following code snippet shows the exceptions that are suppressed:

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

Programming in Java

Catching Multiple Exceptions

- Multi-`catch` statement:
  - Reduces the amount of code to be written
  - Avoids catching generic exceptions
  - Separates type alternatives by vertical bars
  - Alternatives must not have inheritance relationship
- Catching an `Exception` object prevents catching other types of exceptions.
- The following embedded Word document shows how to use the new multi-`catch` clause.

multi-catch

Programming in Java

Activity: MultiCatchExample

Let us see how to catch multiple exceptions in Java.

Declaring Exceptions

- Methods:
  - Use the `throws` clause to throw one or more exceptions
  - Stop executing when exception is generated
  - The exception is thrown to the caller
- The following embedded Word document shows how a method throws an exception instead of handling it.

Throwing exception

Declaring Exceptions (Contd.)

- Exceptions while declaring overridden methods:
  - Declare same exceptions
  - Declare fewer exceptions
  - Declare more specific exceptions
  - Do not declare additional exceptions
  - Do not declare more generic exceptions
- The following embedded Word document shows how a method declare multiple exceptions.

Exceptions

Throwing Exceptions

- `throws` clause:
  - Delays exception handling
  - Can repeatedly throw exception up the call stack
  - Must be handled before it is thrown out of the `main()` method
  - Declaration makes it someone else's job to handle it, as shown in the following code snippet:

```
public static void main(String[] args) {
try {
int data = readByteFromFile();
} catch (IOException e) {
System.out.println(e.getMessage());
}
}
```

Method that declared an exception.

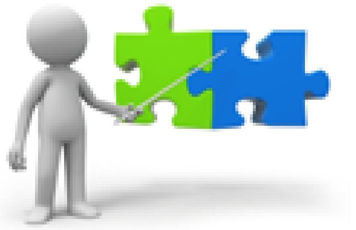
Throwing Exceptions (Contd.)

- Java SE 7 supports rethrowing the precise exception type.
- The following embedded Word document shows how to rethrow an exception in Java SE 7.

Throwing exceptions

Programming in Java

Activity: ThrowExample



Let us see how to rethrow exceptions in Java.

Custom Exceptions

- Custom exceptions:
  - Are created when a class extends the `Exception` class
  - Are not thrown by the standard Java class libraries
  - Class may override methods or add new functionality
  - Capture information about a problem that has occurred
  - Example:

    ```
    throw new DAOException();
    ```
- `getMessage()` method:
  - Used for string type
  - All `Exception` classes inherit it from `Throwable`
  - Returns a string that is stored by exception constructors
- The following embedded Word document shows how to create custom exceptions.

Custom exceptions

Wrapper Exceptions

- Wrapper exception:
  - Hides the type of exception being generated without ignoring it.
  - Example:

```
public class DAOException extends Exception {
public DAOException(Throwable cause) {
super(cause);
}
public DAOException(String message, Throwable cause){
super(message, cause);
}}
```

Wrapper Exceptions (Contd.)

- `Throwable` class:
  - Contains the `getCause()` method to retrieve a wrapped exception
  - Example:

```
    try {

//…
} catch (DAOException e) {
Throwable t = e.getCause();
}
```

Assertions

- Assertion:
  - Ensures that the application is executing as expected
  - Documents and verifies the assumptions and internal logic of a single method
  - Implementation types:
    - Internal invariants
    - Control flow invariants
    - Postconditions and class invariants
  - Combines the exception-handling mechanism with conditionally executed code
  - Syntax:

```
assert <boolean_expression> ;
assert <boolean_expression> :
<detail_expression> ;
```

If `<boolean_expression>` evaluates `false`, then an `AssertionError` is thrown.

Assertions (Contd.)

- Disabled by default
- Enabled or disabled at runtime
- Enabled by using any of the following commands:

  ```
  java -enableassertions MyProgram
  ```

  Or

  ```
  java -ea MyProgram
  ```

- The following code snippet shows the behavior of assertion:

```
if (AssertionsAreEnabled) {
if (condition == false) throw new AssertionError();
}
```

Internal Invariants

- Code without assertion:

```
if (x > 0) {
 // do this
 } else {
 // do that
 }
```

- After implementing assertion:

```
if (x > 0) {
// do this
} else {
assert ( x == 0 );
// do that, unless x is negative
}
```

Here, the assertion evaluates the variable `x` even when it contains the value `0`, which was not handled previously.

Control Flow Invariants

- The following embedded Word document shows an example of control flow invariants.

Control flow variants

Postconditions and Class Invariants

- The following embedded Word document shows an example of postconditions and class invariants.

Postconditions
and class variants

Programming in Java

Quiz

Get Ready for the Challenge

Quiz (Contd.)

- Fill in the blank:
  - The ______________ statement eliminates the need for a lengthy finally block.

- Solution:
  - `try`-with-resources

Quiz (Contd.)

- Which of the following statements is correct regarding exceptions in Java?
  - Custom exceptions are created when a class extends the `RuntimeException`.
  - Checked exceptions must be handled using a `try` or `throws` statement.
  - Unchecked exceptions are discovered by using the `throws` statement.
  - Overridden methods can declare broader set of exceptions.
- Solution:
  - Checked exceptions must be handled using a `try` or `throws` statement.

Summary

- In this session, you learned that:
  - An error should be an exception instead of an expected behavior.
  - The `java.lang.Throwable` class is the parent class of all exceptions.
  - The `try-catch` blocks handle exceptions.
  - The `try`-with-resources statement always closes the opened resources.
  - The new multi-`catch` statement reduces the amount of code to be written.
  - The `throws` clause with a method is used to throw one or more exceptions.
  - Custom exceptions are created when a class extends the `Exception`.
  - Assertion ensures that the application is executing as expected.