**Administrative**

**Today’s session**

Exceptions

**Session Topics**

**Exceptions**

● **Exception handling** enables an application to gracefully respond to an error.

● Exception handling has two parts:

✓ One part listens for and signals an error. This is called **throwing an exception**.

✓ Another part responds to the error. This is called **handling an exception**.

**trystatement**

● Exception handling is typically implemented with the **trystatement**.

● The trystatement includes:

✓ A **try block** that contains code that may generate an exception.

✓ A **catch block** that contains code that is run if an exception occurs in the try block.

● trystatement syntax (version 1):

try

{

<block>

}

catch (<exception-class> <exception-variable>)

{

<block>

}

● If an exception occurs in a try block:

1) An exception object is created containing information about the error.

2) A search is made of catch blocks for an exception class matching the exception object. If there is a matching catch block, then that code executes. If there is no matching catch block, then the application ends with the error.

● Java has a hierarchy of exception classes for handling all kinds of common errors.

● There are two types of exceptions:

✓ **Checked exceptions** are errors that must be handled in an application.

✓ **Unchecked exceptions** are errors that are *not required* to be handled in an application.

● The **Throwable class** is the top of the exception classes tree.

● Here is a subset of the Throwable class:

Throwable

(unchecked exceptions)

Error

AssertionError – assert statement was false.

IOError – hardware error.

VirtualMachineError

InternalError – unexpected (Java Virtual Machine) JVM internal error.

OutOfMemoryError – JVM out of memory.

StackOverflowError – cannot recurse any further.

Exception

(checked exceptions)

CloneNotSupportedException – Cloneable interface not implemented.

IOException

*FileNotFoundException – specified file doesn’t exist.*

(unchecked exceptions)

RuntimeException

ArithmeticException – attempt to divide by zero

ArrayStoreException – attempt to store wrong data type in array.

IllegalArgumentException

*NumberFormatException – attempt to convert string to number*

IndexOutOfBoundsException

ArrayIndexOutOfBoundsException – index is negative or beyond array.

StringIndexOutOfBoundsException – index is negative or beyond string.

NegativeArraySizeException – attempt to create array with negative size.

NoSuchElementException

InputMismatchException – Scanner token is wrong data type.

NullPointerException – attempt to use object variable that is null.

● The **Error exception** class and all its subclasses are unchecked exceptions.

● The **RuntimeException class** and all its subclasses are unchecked exceptions.

● The unchecked exceptions that should be used in your application include those that are triggered by user or file input.

● In a catch block, information about an exception may be obtained using the **getMessage method**.

● See **Exceptions** sample application on Blackboard.

**Custom exception**

● A Java exception class may be extended to create a custom exception class.

● In its simplest form, the custom exception can pass a custom message to the Java exception.

● To create a simple custom exception:

1) Define a class that extends a Java exception:

public class <custom-exception-class> extends <Java-exception-class>

{

private static final long serialVersionUID = 1L; // Java-required field

public <custom-exception-class> (String message)

{

super(message);

}

}

2) Define a method that throws (triggers) the custom exception:

<method-header> (<parameter-list>) throws <custom-exception-class>

{

…

throw new <custom-exception-class> (<argument-list>);

…

}

● A Java exception class throws its own exception.

● A custom exception class is triggered with the **throw statement**.

● throw statement syntax:

throw new <exception-class>(<argument-list>);

● Exceptions are handled in one of two ways:

✓ In a try statement.

✓ In a method that is declared with the **throws** modifier and includes **throw new** statements.

● Generally, its better to use the try statement approach so that the developer can more easily see how an exception is handled.

● See **Exceptions** sample application on Blackboard.

**Multiple catch blocks and Finally block**

● The trystatement may include more than one **catch block** and an optional **finally block**.

● Each catch block traps a different error type.

● The finally block, if specified, runs whether an exception occurred or not.

● trystatement syntax (version 2):

try

{

<block>

}

catch (<exception-class-1> <exception-variable-1>)

{

<block>

}

catch (<exception-class-2> <exception-variable-2>)

{

<block>

}

…

catch (<exception-class-n> <exception-variable-n>)

{

<block>

}

finally

{

<block>

}

● If multiple catch blocks are used, put the most specific exception type first and the most general exception type last.

● See **Exception with multiple catch blocks** sample application on Blackboard.

● The **finally block** may be used to clean up any work performed in the try and catch blocks.

● See **Exception with finally block** sample application on Blackboard.

● Many errors occur during application development. We typically resolve those so exception handling is **not** needed.

● Exception handling that should be incorporated in a final application version includes:

✓ Checked exceptions (FileNotFoundException, CloneNotSupportedException, etc.).

✓ Exceptions caused by bad user input (NumberFormatException , ArithmeticException, etc.).