Exception Handling

About This Lecture

- Purpose
  - To introduce exception handling mechanism for structured error recovery
- What You Will Learn
  - Classes Throwable, Exception, Error
  - Throw and catch exceptions
  - Declare custom exceptions
  - *Finally* block for cleanup

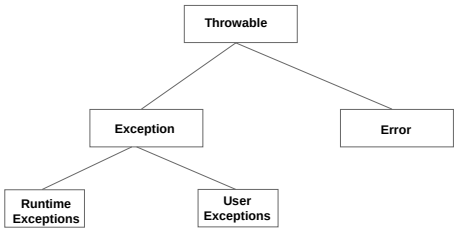
Conventional Approach

- Use return value to report errors
  - Method may have void return type
  - It may be hard to find reasonable error value
  - Clumsy in programming: exhaustive testing of return value after each method call
  - Error processing is usually done by immediate caller

Exception Handling

- Introduced by Ada, adopted by C++ and Java
- Fix the previous problems
- It has overhead in execution time and memory resources

Exception Classification



Class Throwable

- Super class of all Exceptions and Errors
- Implements
  - String getMessage() // detailed message
  - String toString() // short description
  - String printStackTrace() // trace information
  - Throwable()
  - Throwable(String)

Exception Types (1/3)

- Runtime exceptions, usually reported by Java VM runtime system
  - ArithmeticException
  - ArrayIndexOutOfBoundsException
  - ArrayStoreException
  - ClassCastException
  - IllegalArgumentException
  - IllegalMonitorStateException
  - IllegalThreadStateException
  - IndexOutOfBoundsException
  - NegativeArraySizeException

Exception Types (2/3)

- User defined exceptions
  - Any class inheriting class java.lang.Exception
  - Use meaningful name
  - Use instance data members and constructors to pass more exception information

```
public class NegativeValue extends Exception {
  public double value;
  public NegativeValue(double v) { value = v; }
  public NegativeValue (double v, String s) {
   super(s);
   value = v;
  }
```


Exception Types (3/3)

- If a method body may throw a user exception, directly or indirectly, that is not processed in the same method, the method must declare this fact with ***throws*** clause

```
void f() throws Exception1, Exception2 { …….}
```

- Runtime exceptions don't need to be declared for methods

Errors

- Errors are serious exceptions that usually cannot be recovered
  - LinkageError
  - VirtualMachineError
  - AWTError
  - ……
- Errors are not to be caught

Throw an Exception

- When code detects an error, throw an object of the corresponding exception type

```
public class NegativeValue extends Exception {}

void deposit(double x)  throws NegativeValue {
    if (x < 0) throw new NegativeValue();
    //  normal processing
}
```

- If *throw* is executed, code after the *throw* will not be executed

Try and Catch Blocks (1/4)

- If a segment of code potentially throws exceptions, enclose it in a try block followed by one or more catch blocks

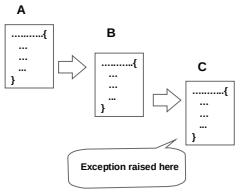
```
void f(double x) {
   try {
      deposit(x);
   }
   catch (NegativeValue e) {
      System.out.println(e);
   }
}
```

Try and Catch Blocks (2/4)

- If no exception happens, after executing body of try block, execute the first line after the catch blocks
- If exception happens, code in try block after the excepting firing line is skipped, start search for exception handler
- First sequentially try each catch blocks. If one of them has a matching exception type, the body of first such one will be executed, with the exception object as a parameter; then execution continues at the first line after the catch blocks

Try and Catch Blocks (3/4)

- If exception happens but no catch block matches the exception type, current method terminates, exception object is thrown at the invocation line of the caller method. Repeat this process



Runtime system looks first in C for a handler and executes it if found

If not, it looks in B and executes it if found

If not, it looks in A and executes it, if found

If not, runtime system raises error, prints stack trace, and terminates the program

Try and Catch Blocks (4/4)

- If the body of main() throws an exception, the application terminates execution
- Exception handling is expensive in terms of memory space and run-time. Avoid throwing exception objects that will be caught in the same method

Multiple Catch Blocks

- Order the catch blocks from more specific type to more general

```
try {
  ...
}
catch (ArithmeticException ae) {
  …
}
catch (NumberFormatException nfe) {
  ...
}
catch (Exception e) {…}
```

Will catch every exception, as Exception is the superclass of all exceptions

Rethrow Exception Objects

- Rethrowing an exception (same or different) in a catch block gives the caller method a chance to process it

```
try {
  ……
}
Catch (TheException e) {
  perform local processing
  throw e;  // To be caught by caller
}
```


Finally Clause (1/3)

- ***Finally clause*** is executed in all cases

```
int avg = 0;
try {
  avg = sum / n;
  System.out.println(…….);
}
catch (Exception e) {
  System.out.println(…..);
}
finally {
  // release some resources, or do any common task
}
```

Finally Clause (2/3)

- With a finally clause, catch blocks are optional
- Finally clause is mainly used to release resources acquired before the try/catch blocks
- If no exception happens, first execute try body, then execute finally clause, then the code after the try/catch/finally structure
- If exception happens, and it is not caught by one of the following catch blocks, then skip the code of try block after the exception source, execute the finally clause, terminate the method call; the exception is thrown in the caller method on the invocation line

Finally Clause (3/3)

- If exception happens, and it is caught by one of the following catch blocks, then skip the code of try block after the exception source
  - If the catch block doesn't rethrow exception, execute the body of the catch block, then the finally clause, then the code after the try/catch/finally structure
  - If the catch block does rethrow exception, execute the body of the catch block up to the rethrow, then the finally clause, then terminate the method call; the exception is thrown in the caller method on the invocation line

Nesting Try/Catch Structures

- Try/catch structure can be nested

```
try {
    …
}
catch (ArithmeticException ae) {
    …
    try {
        …
    }
    catch (IOException ioe) {…}
}
catch (Exception e) {…}
```