5. Exception Handling and Debugging

Introduction

An exception is an error occurred during a program execution so that the normal flow of program execution is interrupted and the completion of the program execution is prevented. A program user might expect some types of errors:

- User input errors
- Device errors
- Coding errors

Java environment provides exceptions and errors handling classes and methods that can help programmer to write programs that are able to handle such problems as a bad type casting, out-of-bound array accessing errors, a read past the end of file, and so on. When there is a run time error, a program should

- Inform the user of errors or exceptions
- Save the necessary information and return to a safer state
- Exit the program gracefully

When programs are not working properly, debuggers of an integrated development tools in such Java program developing environments as JBuilder, Symantec Cafe, or Sun's Forte can be helpful for trouble shooting. However, if you are using a new version of Java that is not yet supported by any developing environment, basic debugging methods that uses print statements to observe values of variables, sates of objects, and so on may be needed to help you correct problems.

In addition to program debugging, it is also necessary to document your Java classes to help your class users to minimizing their programming errors. API documentation can be found at:

http://java.sun.com/j2se/1.3/docs/api/index.html

Exception Handling

- Exceptions are mechanisms for isolating errors or unusual situation from the code from a regular program execution.
- Java keywords as a listed below can be used in exception handling blocks:
    - try
    - catch
    - finally
    - throw
    - throws
- A **throws** clause must be included in a method definition if any uncaught exceptions may occur while the method is running
- Exception Definition:
    - Checked and unchecked exceptions
    - Checked exceptions must be listed in the throws clause
    - Unchecked exceptions are intended only for severe, unpredictable errors that could happen in any method
        - Examples of unchecked exceptions:
            - NullPointerException
            - ArrayIndexOutOfBoundException
    - Synchronous, triggered by a specific line of code and not from an outside event

    - Events that alter a thread's normal path of execution
    - Can be detected by either software or hardware

- java.lang contains the core classes Exception, RuntimeException, and Error
- Exception handling process involves "**call stack**" and includes
    - Throwing an Exception - an act of detecting an abnormal condition and generating an exception
    - Catching the exception - exception handler takes control
    - Passing control back to the program
- Error handling codes include in the following blocks
    - try blocks
    - catch blocks
    - finally block (optional)
- Three options
    - Use a catch clause to catch an exception and handle it
    - Omit a catch clause for an exception and let the call stack unwind further looking for a handler in an outer try block
    - Catch an exception and then re-throw it

Try Blocks

```
try {
        . . .
    }
catch (ExceptionType ref)
    {
        . . .
     }
```

Example:

```
try {
        input = (char) System.in,read();
    }
catch (Exception e)  // Catch all exceptions
    {
    System.out.println("Error: " e.toString());
     }
```

Finally Block

- Provide code that always executed regardless of whether or not an exception occurs
- An ideal place to place code that release resources to prevent "resource leaks"

```
try
  {
   statements;
   resource-requesting statements;
  }
```

```
catch(AkindOfException ex1)
 {
exception handling statements;
 }
catch(AnotherKindOfException ex2)
 {
exception handling statements;
}
finally
 {
  statements;
  resource-releasing statements;
 }
```

java.lang.Exception

public class Exception extends Throwable

Java exception handling classes allows a program to catch all types of exceptions:

- ClassNotFoundException
- CloneNotSupportedException
- IllegalAccessException
- InterruptedException
- NoSuchFieldException
- NoSuchMethodException
- RuntimeException
    - ArithmeticException: Divide by Zero
    - ArrayStoreException

    - ClassCastException
    - IllegalArgumentsException
        - IllegalThreadStateException
        - NumberFormatException
    - IllegalMonitorStateException
    - IllegalStateException
    - IndexOutOfBoundsException
        - ArrayIndexOutOfBoundsException
        - StringIndexOutOfBoundsException
    - NegativeArraySizeException
    - NullPointerException
    - SecurityException

java.lang package Errors

- LinkageError
    - ClassCircularityError
    - ClassFormatError
    - ExceptionInInitializerError
    - IncompatibleClassChangeError
        - AbstractMethodError
        - IllegalAccessError
        - InstantiationError
        - NoSuchFieldError
        - NoSuchMethodError
    - NoClassDefFoundError
    - UnsatisfiedLinkError
    - VerifyError
- ThreadDeath
- VirtualMachineError (Abstract class)

    - InternalError
    - OutOfMemoryError
    - StackOverflowError
    - UnknownError
- AWTError

java.util package Exceptions

Exception

- RuntimeException
    - EmptyStackException
    - MissingResourceException
    - NoSuchElementsException
- TooManyListenersException

java.io package Exceptions

Exception

- IOException
    - CharConversionException
    - EOFException
    - FileNotFoundException
    - InterruptedIOException
    - ObjectStreamException
        - InvalidClassException
        - InvalidObjectException
        - NotActiveException
        - NotSerializableException
        - OptionalDataException
        - StreamCorruptedException
        - WriteAbortedException
    - SyncFailedException

- UnsupportedCodingException
- UTFDataFormatException

java.awt package Exceptions

Exception

- AWTException
- RuntimeException
  - IllegalStateException
  - IllegalComponentsStateException

java.net package Exceptions

Exception

IOException

BindException

MalformedURLException

ProtocolException

SocketException

ConnectException

NoRouteToHostException

UnknownHostException

UnknownServiceException

Example 5-1:

```
// DateStack.java
import java.util.*;
class DateStack
{
   public static void main(String args[])
  {
      //Create a Stack object
      Stack DTStack = new Stack();
      // Create Date objects and push on the top of the stack
      Date nowT1 = new Date();
      // Push Date objects onto the stack
      DTStack.push(nowT1);
      System.out.println("push(nowT1): " + nowT1);
      Date nowT2 = new Date();
      DTStack.push(nowT2);
      System.out.println("push(nowT2): " + nowT2);
      Date nowT3 = new Date();
      DTStack.push(nowT3);
      System.out.println("push(nowT3): " + nowT3);
      // peek at the Stack
      System.out.println("peek(TOP):   " + DTStack.peek());
      // Pop until the stack is empty
      int stackPtr = 0;
      while(!DTStack.empty())
       {
         System.out.println("pop(" + stackPtr + "):      "+
             TStack.pop());
         stackPtr++;
       }
     }
}
```

```
E:\LinJavaExs\4_ArrayString>jdb DateStack
Initializing jdb...
> run DateStack
run  DateStack
Java HotSpot(TM) Client VM warning: Setting of property "java.compiler"
is ignor
ed
>
VM Started: push(nowT1): Tue Jun 05 14:06:04 EST 2001
push(nowT2): Tue Jun 05 14:06:06 EST 2001
push(nowT3): Tue Jun 05 14:06:06 EST 2001
peek(TOP):   Tue Jun 05 14:06:06 EST 2001
pop(0):      Tue Jun 05 14:06:06 EST 2001
pop(1):      Tue Jun 05 14:06:06 EST 2001
pop(2):      Tue Jun 05 14:06:04 EST 2001

The application exited
```

Example 5-2:

```
/*
  StackExceptionTest.java
*/
import java.util.*;
public class StackExceptionTest
{
   public static void main(String[] args)
   {
      int n = 0;
      int numTries = 1000000;
      Stack stack_1 = new Stack();
      long t1, t2;
      // Empty Stack Testing
      System.out.println("Empty Stack Testing");
      t1 = new Date().getTime();
      while(n++ <= numTries)
        {
         if (!stack_1.empty()) stack_1.pop();
        }
      t2 = new Date().getTime();
      System.out.println((t2 - t1) + " milliseconds");

      // Pop an empty stack
      // Catch resulting exception
      System.out.println("Catching EmptyStackException");
      t1 = new Date().getTime();
      while(n++ <= numTries)
      {
         try
         {
            stack_1.pop();
         }
         catch(EmptyStackException e)
         {
         }
      }
      t2 = new Date().getTime();
      System.out.println((t2 - t1) + " milliseconds");
   }
}
```

Result:

```
Empty Stack Testing
110 milliseconds
Catching EmptyStackException
0 milliseconds
Press any key to continue...
```

Testing and Debugging Techniques

- Test case selection and evaluation
- Document program assertion
- Effective Testing
    - Select test cases that provide complete coverage
    - Unit test (testing in isolation)
        - Use main() to call new functions with parameters obtained from
            - user input -- manual testing case generation
            - a function that randomly generates parameter values -- automatic test case generation
            - a file
        - Pick inputs that have known results - validate outputs
        - Compare results
    - Program traces (print results at certain known points)
        - Trace over/inspect
        - Trace into
        - Setting breakpoints
        - Watch values of variables
        - Print states (all instance variables) of running objects
    - Put test code in main() method for isolated unit testing
    - Use printStackTrace() of Throwable class
    - Thread.dumpStak()

Java Debugger

The jdb is a text-based, command-line-oriented debugger for Java classes. The jdb runs within a Java interpreter.

Help Menu

```
c:\jdb.exe
Initializing jdb...
> help
** command list **
run [class [args]]        -- start execution of application's main class

threads [threadgroup]     -- list threads
thread <thread id>        -- set default thread
suspend [thread id(s)]    -- suspend threads (default: all)
resume [thread id(s)]     -- resume threads (default: all)
where [thread id] | all   -- dump a thread's stack
wherei [thread id] | all  -- dump a thread's stack, with pc info
up [n frames]             -- move up a thread's stack
down [n frames]           -- move down a thread's stack
kill <thread> <expr>      -- kill a thread with the given exception object
interrupt <thread>        -- interrupt a thread

print <expr>              -- print value of expression
dump <expr>               -- print all object information
eval <expr>               -- evaluate expression (same as print)
set <lvalue> = <expr>     -- assign new value to field/variable/array element
locals                    -- print all local variables in current stack frame

classes                   -- list currently known classes
class <class id>          -- show details of named class
methods <class id>        -- list a class's methods
fields <class id>         -- list a class's fields

threadgroups              -- list threadgroups
threadgroup <name>        -- set current threadgroup

stop in <class id>.<method>[(argument_type,...)]
                          -- set a breakpoint in a method
stop at <class id>:<line> -- set a breakpoint at a line
clear <class id>.<method>[(argument_type,...)]
                          -- clear a breakpoint in a method
clear <class id>:<line>   -- clear a breakpoint at a line
clear                     -- list breakpoints
catch <class id>          -- break when specified exception thrown
ignore <class id>         -- cancel 'catch'  for the specified exception
```

```
watch [access|all] <class id>.<field name>
                          -- watch access/modifications to a field
unwatch [access|all] <class id>.<field name>
                          -- discontinue watching access/modifications to a field
trace methods [thread]    -- trace method entry and exit
untrace methods [thread]  -- stop tracing method entry and exit
step                      -- execute current line
step up                   -- execute until the current method returns to its caller
stepi                     -- execute current instruction
next                      -- step one line (step OVER calls)
cont                      -- continue execution from breakpoint

list [line number|method] -- print source code
use (or sourcepath) [source file path]
                          -- display or change the source path
exclude [class id ... | "none"]
                          -- do not report step or method events for specified
classes
classpath                 -- print classpath info from target VM

monitor <command>         -- execute command each time the program stops
monitor                   -- list monitors
unmonitor <monitor#>      -- delete a monitor
read <filename>           -- read and execute a command file

lock <expr>               -- print lock info for an object
threadlocks [thread id]   -- print lock info for a thread

disablegc <expr>          -- prevent garbage collection of an object
enablegc <expr>           -- permit garbage collection of an object

!!                        -- repeat last command
<n> <command>             -- repeat command n times
help (or ?)               -- list commands
version                   -- print version information
exit (or quit)            -- exit debugger

<class id>: full class name with package qualifiers or a
pattern with a leading or trailing wildcard ('*').
<thread id>: thread number as reported in the 'threads' command
<expr>: a Java(tm) Programming Language expression.
Most common syntax is supported.

Startup commands can be placed in either "jdb.ini" or ".jdbrc"
in user.home or user.dir
```