Exceptions:
When something goes wrong

Image from Wikipedia

Conditions that cause exceptions

> Error internal to the Java Virtual Machine

> “Standard” exceptions:
    Divide by zero
    Array index out of bounds
    Etc.

> Manually generated exceptions - using a throw statement

Exception handling fundamentals

> Exceptions are represented by classes

> All exceptions are derived from a class named Throwable

> Two direct subclasses of Throwable:
    Error – exceptions from the Java Virtual Machine
    Exception – handles errors from program activity
        Subclasses of exception handle various errors

> Standard package java.lang defines several exception sub-classes

> Custom exception classes are frequently created

Exception related keywords

Five keywords:

try

catch

throw

throws

finally

Using try and catch

try and catch go together

```
try  {
     // block of code to monitor for errors
}

catch (ExceptType1 exOb) {
     // handler for ExceptType1
}

catch (ExceptType2 exOb) {
     // handler for ExceptType2
}
   Etc.
```

> Where *ExceptType* is the type of exception that will be caught.

> exOb is a name you give to the exception object.

Demonstrate exception handling.

```
class ExcDemo1 {
    public static void main(String args[]) {
    int nums[] = new int[4];

    try {
        System.out.println("Before exception is generated.");
                // Generate an index out-of-bounds exception.
        nums[7] = 10;
        System.out.println("this won't be displayed");
    }

    catch (ArrayIndexOutOfBoundsException exc) {
                // catch the exception
        System.out.println("Index out-of-bounds!");
    }

    System.out.println("After catch statement.");
  }
}
```

```
class ExcDemo1 {
    public static void main(String args[]) {
    int nums[] = new int[4];

    try {
        System.out.println("Before exception is generated.");
        Nums[7] = 10;        // Generate an index out-of-bounds exception.
        System.out.println("this won't be displayed");
    }

    catch (ArrayIndexOutOfBoundsException exc) {
              // catch the exception
        System.out.println("Index out-of-bounds!");
    }

    System.out.println("After catch statement.");
  }
}
```

OUTPUT

Before exception is generated.
Index out-of-bounds!
After catch statement

```
/* An exception can be generated by one
   method and caught by another. */

class ExcTest {
      // Generate an exception.
    static void genException() {
        int nums[] = new int[4];
        System.out.println("Before exception is generated.");

         // generate an index out-of-bounds exception
        nums[7] = 10;
        System.out.println("this won't be displayed");
  }
}

class ExcDemo2 {
    public static void main(String args[]) {

    try {
          ExcTest.genException();
    }
    catch (ArrayIndexOutOfBoundsException exc) {
          // catch the exception
          System.out.println("Index out-of-bounds!");
    }
    System.out.println("After catch statement.");
  }
}
```

What if an exception is uncaught?
(by your program)

What if an exception is uncaught?
(by your program)

> The JVM default handler will catch it.

> Then what?

What if an exception is uncaught?
(by your program)

> The JVM default handler will catch it.

> Then what?

> Program execution terminates
JVM displays a stack trace
and error message

Another example

```
class ExcTypeMismatch {
  public static void main(String args[]) {
      int nums[] = new int[4];

      try {
         System.out.println("Before exception is generated.");

               // generate an index out-of-bounds exception
         nums[7] = 10;
         System.out.println("this won't be displayed");
       }

       catch (ArithmeticException exc) {
               // catch the exception
         System.out.println("Index out-of-bounds!");
       }

      System.out.println("After catch statement.");
  }
}
```

(what happens when this executes?)

Graceful error handling

```
class ExcDemo3 {
  public static void main(String args[]) {
      int numer[] = { 4, 8, 16, 32, 64, 128 };
      int denom[] = { 2, 0, 4, 4, 0, 8 };

      for(int i=0; i<numer.length; i++) {
          try {
             System.out.println(numer[i] + " / " +
                        denom[i] + " is " +
                        numer[i]/denom[i]);
          }
          catch (ArithmeticException exc) {
              // catch the exception
          System.out.println("Can't divide by zero!");
          }
      }
  }
}
```

Output

```
4 / 2 is 2
Can’t divide by zero!
16 / 4 is 4
32 / 4 is 8
Can’t divide by zero!
128 / 8 is 16
```

Multiple catch statements

```
class ExcDemo4 {
  public static void main(String args[]) {
    // Here, numer is longer than denom.
    int numer[] = { 4, 8, 16, 32, 64, 128, 256, 512 };
    int denom[] = { 2, 0, 4, 4, 0, 8 };

    for(int i=0; i<numer.length; i++) {
      try {
        System.out.println(numer[ i ] + " / " +denom[ i ] + " is " +
                    numer[ i ]/denom[ i ]);
      }
      catch (ArithmeticException exc) {
        System.out.println("Can't divide by Zero!");
      }
      catch (ArrayIndexOutOfBoundsException exc) {
        System.out.println("No matching element found.");
      }
    }
  }
}
```

Output

```
4 / 2 is 2
Can’t divide by zero!
16 / 4 is 4
32 / 4 is 8
Can’t divide by zero!
128 / 8 is 16
No matching element found.
No matching element found.
```

Subclasses must precede superclasses in catch statements.

```
class ExcDemo5 {
  public static void main(String args[]) {
    // Here, numer is longer than denom.
    int numer[] = { 4, 8, 16, 32, 64, 128, 256, 512 };
    int denom[] = { 2, 0, 4, 4, 0, 8 };

    for(int i=0; i<numer.length; i++) {
      try {
        System.out.println(numer[i] + " / " +
                           denom[i] + " is " +
                           numer[i]/denom[i]);
      }
      catch (ArrayIndexOutOfBoundsException exc) {
        // catch the exception
        System.out.println("No matching element found.");
      }
      catch (Throwable exc) {
        System.out.println("Some exception occurred.");
      }
    }
  }
}
```

```
// Use a nested try block.
class NestTrys {
  public static void main(String args[]) {
    int numer[] = { 4, 8, 16, 32, 64, 128, 256, 512 };
    int denom[] = { 2, 0, 4, 4, 0, 8 };        // denom shorter than numer

    try {          // outer try
        for(int i=0; i<numer.length; i++) {
            try {         // nested try
               System.out.println(numer[i] + " / " +
                           denom[i] + " is " +
                           numer[i]/denom[i]);
            }
            catch (ArithmeticException exc) {       // nested catch
                System.out.println("Can't divide by Zero!");
            }
        }
    }
    catch (ArrayIndexOutOfBoundsException exc) {
        System.out.println("No matching element found.");
        System.out.println("Fatal error -- program terminated.");
    }
  }
}
```

How to manually throw an exception.

```
class ThrowDemo {
  public static void main(String args[]) {
    try {
      System.out.println("Before throw.");
      throw new ArithmeticException();
    }
    catch (ArithmeticException exc) {
      // catch the exception
      System.out.println("Exception caught.");
    }
    System.out.println("After try/catch block.");
  }
}
```

Recall: exceptions are objects

Using the throws keyword

- If a method generates an exception that it doesn't handle, it must declare that exception in a throws clause:

  ret-type methName(*param-list*) throws *except-list* {
      // body of a method or class
  }

- Exceptions that are subclasses of Error or RuntimeException don't need to be specified in a throws list. All other types do.

```
class ThrowsDemo {
   public static char prompt(String str)
      throws java.io.IOException {

      System.out.print(str + ": ");
      return (char) System.in.read();
   }

   public static void main(String args[]) {
      char ch;

      try {
         ch = prompt("Enter a letter");
      }
      catch(java.io.IOException exc) {
          System.out.println("I/O exception occurred.");
          ch = 'X';
      }

      System.out.println("You pressed " + ch);
    }
}
```

Possible error

Possible error

Exception	Meaning
ArithmeticException	Arithmetic error, such as integer divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBoundsException	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

Table 9-2 The Unchecked Exceptions Defined in **java.lang**

Exception	Meaning
ArithmeticException	Arithmetic error, such as integer divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.

IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBoundsException	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

Table 9-2 The Unchecked Exceptions Defined in **java.lang**

Custom exceptions can be created

```
class NonIntResultException extends Exception {
    int n;
    int d;                          // new exception name

    NonIntResultException(int i, int j) {       // constructor
       n = i;
       d = j;
    }

    public String toString( ) {
       return "Result of " + n + " / " + d + " is non-integer.";
    }
}
```

```
class CustomExceptDemo {
  public static void main(String args[]) {
    int numer[ ] = { 4, 8, 15, 32, 64, 127, 256, 512 };
    int denom[ ] = { 2, 0, 4, 4, 0, 8 };

    for(int i=0; i<numer.length; i++) {
      try {
        if((numer[i]%2) != 0)
          throw new NonIntResultException(numer[i], denom[i]);

        System.out.println(numer[i] + " / " + denom[i] + " is " +
                                          numer[i]/denom[i]);
      }
      catch (ArithmeticException exc) {
          System.out.println("Can't divide by Zero!");
      }
      catch (ArrayIndexOutOfBoundsException exc) {
          System.out.println("No matching element found.");
      }
      catch (NonIntResultException exc) {
          System.out.println(exc);
      }
    }
  }
}
```

There is also a set of exceptions that are checked (i.e. caught) by default. They are defined in java.lang

see Table 9-3

Using the keyword finally

Defines a block of code that will execute when a try or catch block is executed.