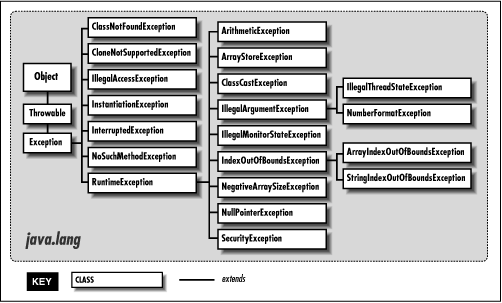
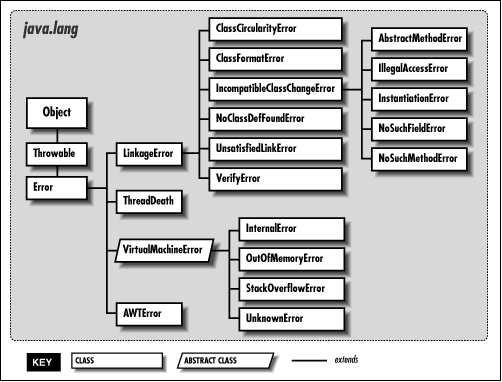
**The Exception Hierarchy**

The possible exceptions in a Java program are organized in a hierarchy of exception classes. The Throwable class, which is an immediate subclass of Object, is at the root of the exception hierarchy. Throwable has two

**Figure 9.1: Standard Java exception classes**



**Figure 9.2: Standard Java error classes**



**Exceptions**

All of the subclasses of Exception represent exceptional conditions that a normal Java program may want to handle. Many of the standard exceptions are also subclasses of RuntimeException. Runtime exceptions represent runtime conditions that can generally occur in any Java method, so a method is not required to declare that it throws any of the runtime exceptions. However, if a method can throw any of the other standard exceptions, it must declare them in its throws clause.

A Java program should try to handle all of the standard exception classes, since they represent routine abnormal conditions that should be anticipated and caught to prevent program termination.

**Runtime exceptions**

The java.lang package defines the following standard runtime exception classes:

ArithmeticException

This exception is thrown to indicate an exceptional arithmetic condition, such as integer division by zero.

ArrayIndexOutOfBoundsException

This exception is thrown when an out-of-range index is detected by an array object. An out-of-range index occurs when the index is less than zero or greater than or equal to the size of the array.

ArrayStoreException

This exception is thrown when there is an attempt to store a value in an array element that is incompatible with the type of the array.

ClassCastException

This exception is thrown when there is an attempt to cast a reference to an object to an inappropriate type.

IllegalArgumentException

This exception is thrown to indicate that an illegal argument has been passed to a method.

IllegalMonitorStateException

This exception is thrown when an object's wait(), notify(), or notifyAll() method is called from a thread that does not own the object's monitor.

IllegalStateException

This exception is thrown to indicate that a method has been invoked when the run-time environment is in an inappropriate state for the requested operation. This exception is new in Java 1.1.

IllegalThreadStateException

This exception is thrown to indicate an attempt to perform an operation on a thread that is not legal for the thread's current state, such as attempting to resume a dead thread.

IndexOutOfBoundsException

The appropriate subclass of this exception (i.e., ArrayIndexOutOfBoundsException or StringIndexOutOfBoundsException) is thrown when an array or string index is out of bounds.

NegativeArraySizeException

This exception is thrown in response to an attempt to create an array with a negative size.

NullPointerException

This exception is thrown when there is an attempt to access an object through a null object reference. This can occur when there is an attempt to access an instance variable or call a method through a null object or when there is an attempt to subscript an array with a nullobject.

NumberFormatException

This exception is thrown to indicate that an attempt to parse numeric information in a string has failed.

RuntimeException

The appropriate subclass of this exception is thrown in response to a runtime error detected at the virtual machine level. Because these exceptions are so common, methods that can throw objects that are instances of RuntimeException or one of its subclasses are not required to declare that fact in their throws clauses.

SecurityException

This exception is thrown in response to an attempt to perform an operation that violates the security policy implemented by the installedSecurityManager object.

StringIndexOutOfBoundsException

This exception is thrown when a String or StringBuffer object detects an out-of-range index. An out-of-range index occurs when the index is less than zero or greater than or equal to the length of the string.

**Other exceptions**

The java.lang package defines the following standard exception classes that are not runtime exceptions:

ClassNotFoundException

This exception is thrown to indicate that a class that is to be loaded cannot be found.

CloneNotSupportedException

This exception is thrown when the clone() method has been called for an object that does not implement the Cloneable interface and thus cannot be cloned.

Exception

The appropriate subclass of this exception is thrown in response to an error detected at the virtual machine level. If a program defines its own exception classes, they should be subclasses of the Exception class.

IllegalAccessException

This exception is thrown when a program tries to dynamically load a class (i.e., uses the forName() method of the Class class, or thefindSystemClass() or the loadClass() method of the ClassLoader class) and the currently executing method does not have access to the specified class because it is in another package and not public. This exception is also thrown when a program tries to create an instance of a class (i.e., uses the newInstance() method of the Class class) that does not have a zero-argument constructor accessible to the caller.

InstantiationException

This exception is thrown in response to an attempt to instantiate an abstract class or an interface using the newInstance() method of theClass class.

InterruptedException

This exception is thrown to signal that a thread that is sleeping, waiting, or otherwise paused has been interrupted by another thread.

NoSuchFieldException

This exception is thrown when a specified variable cannot be found. This exception is new in Java 1.1.

NoSuchMethodException

This exception is thrown when a specified method cannot be found.

**Errors**

The subclasses of Error represent errors that are normally thrown by the class loader, the virtual machine, or other support code. Application-specific code should not normally throw any of these standard error classes. If a method does throw an Error class or any of its subclasses, the method is not required to declare that fact in its throws clause.

A Java program should not try to handle the standard error classes. Most of these error classes represent non-recoverable errors and as such, they cause the Java runtime system to print an error message and terminate program execution.

The java.lang package defines the following standard error classes:

AbstractMethodError

This error is thrown in response to an attempt to invoke an abstract method.

ClassCircularityError

This error is thrown when a circular reference among classes is detected during class initialization.

ClassFormatError

This error is thrown when an error is detected in the format of a file that contains a class definition.

Error

The appropriate subclass of this error is thrown when an unpredictable error, such as running out of memory, occurs. Because of the unpredictable nature of these errors, methods that can throw objects that are instances of Error or one of its subclasses are not required to declare that fact in their throws clauses.

ExceptionInInitializerError

This error is thrown when an unexpected exception is thrown in a static initializer. This error is new in Java 1.1.

IllegalAccessError

This error is thrown when a class attempts to access a field or call a method it does not have access to. Usually this error is caught by the compiler; this error can occur at run-time if the definition of a class changes after the class that references it was last compiled.

IncompatibleClassChangeError

This error or one of its subclasses is thrown when a class refers to another class in an incompatible way. This situation occurs when the current definition of the referenced class is incompatible with the definition of the class that was found when the referring class was compiled. For example, say class A refers to a method in class B. Then, after class A is compiled, the method is removed from class B. When class A is loaded, the run-time system discovers that the method in class B no longer exists and throws an error.

InstantiationError

This error is thrown in response to an attempt to instantiate an abstract class or an interface. Usually this error is caught by the compiler; this error can occur at run-time if the definition of a class is changed after the class that references it was last compiled.

InternalError

This error is thrown to signal an internal error within the virtual machine.

LinkageError

The appropriate subclass of this error is thrown when there is a problem resolving a reference to a class. Reasons for this may include a difficulty in finding the definition of the class or an incompatibility between the current definition and the expected definition of the class.

NoClassDefFoundError

This error is thrown when the definition of a class cannot be found.

NoSuchFieldError

This error is thrown in response to an attempt to reference an instance or class variable that is not defined in the current definition of a class. Usually this error is caught by the compiler; this error can occur at run-time if the definition of a class is changed after the class that references it was last compiled.

NoSuchMethodError

This error is thrown in response to an attempt to reference a method that is not defined in the current definition of a class. Usually this error is caught by the compiler; this error can occur at run-time if the definition of a class is changed after the class that references it was last compiled.

OutOfMemoryError

This error is thrown when an attempt to allocate memory fails.

StackOverflowError

This error is thrown when a stack overflow error occurs within the virtual machine.

ThreadDeath

This error is thrown by the stop() method of a Thread object to kill the thread. Catching ThreadDeath objects is not recommended. If it is necessary to catch a ThreadDeath object, it is important to re-throw the object so that it is possible to cleanly stop the catching thread.

UnknownError

This error is thrown when an error of unknown origins is detected in the run-time system.

UnsatisfiedLinkError

This error is thrown when the implementation of a native method cannot be found.

VerifyError

This error is thrown when the byte-code verifier detects that a class file, though well-formed, contains some sort of internal inconsistency or security problem.

VirtualMachineError

The appropriate subclass of this error is thrown to indicate that the Java virtual machine has encountered an error.