Exception Handling

- What are Exceptions ?
- The traditional approach
- Exception handing in Java
- Standard exceptions in Java
- Multiple catch handlers
- Catching multiple exceptions
- finally block
- Try-with-resources statement
- Checked vs unchecked exception
- Stack trace
- Assertion
- jconsole

What are Exceptions ?

- We don't live in a perfect world.
  - there are so many unexpected situations that cause our programs to behave different from our expectations.
- Exceptions refer to
  - situations where programs cannot behave normally.
  - any abnormal situation can lead to a incorrect program behavior.

- Examples of exceptions
  - Happen to divide integer by zero ➔ it may halt the program
  - Non numeric character is entered when an integer is expected.
  - No available memory space when memory allocation is requested by new.
  - No such file when a program tries to open a file.

The basic approach to exceptions

- To prevent a program from abnormal behavior, we need to check every statement that could cause exceptions.
- What exceptions should be considered in the following code ?

```
double badCode(int a[], int y) {
    x = a[y] ;
    z = a[x] ;
    return 1 / z ;
}
```

  - The array *a* may point to null.
  - The index of *y* may not in the valid range: 0 .. a.length-1
  - The index of *x* may not in the valid range: 0 .. a.length-1
  - The value of *z* may be zero, which cannot be accepted.

The traditional approach to exception handling

- Check and handle each exception by if statement.

```
double badCode(int a[], int y) {
    if ( a == null ) { System.out.println("null array") ; return -1.0; }
    if (y < 0 || y >= a.length) {
        System.out.printf("Array index %d is not valid%n", y); return -2.0 ;
    }
    int x = a[y] ;
    if (x < 0 || x >= a.length) {
        System.out.printf("Array index %d is not valid%n", x); return -3.0 ;
    }
    int z = a[x] ;
    if (z == 0) {
        System.out.println("Error: Denominator is 0"); return -4.0 ;
    }
    return 1 / z ;
}
```

Problems with the traditional approach

- Problem #1: Too much overhead for the exception handling codes
- Problem #2: exception handling codes are interspersed with normal codes.
- Problem #3: artificial codes are used to notify exceptional situations.

```
double badCode(int a[], int y) {
    if ( a == null ) { System.out.println("null array") ; return -1.0; }
    if (y < 0 || y >= a.length) {
        System.out.printf("Array index %d is not valid%n", y); return -2.0 ;
    }
    int x = a[y] ;
    if (x < 0 || x >= a.length) {
        System.out.printf("Array index %d is not valid%n", x); return -3.0 ;
    }
    int z = a[x] ;
    if (z == 0) {
        System.out.println("Error: Denominator is 0"); return -4.0 ;
    }
    return 1 / z ;
}
```

Exception Handling in Java

- With try/catch block, we can separate the error checking codes from the normal code.
- Exception handling in Java is similar to that in C++.

```
statements;
try {
    error-prone code...;
}
catch (Exception-type1 e1) {
    code for dealing with e1 exception
}
catch (Exception-type2 e2) {
    code for dealing with e2 exception
}
more-statements;
```

Try block wraps the error-prone code

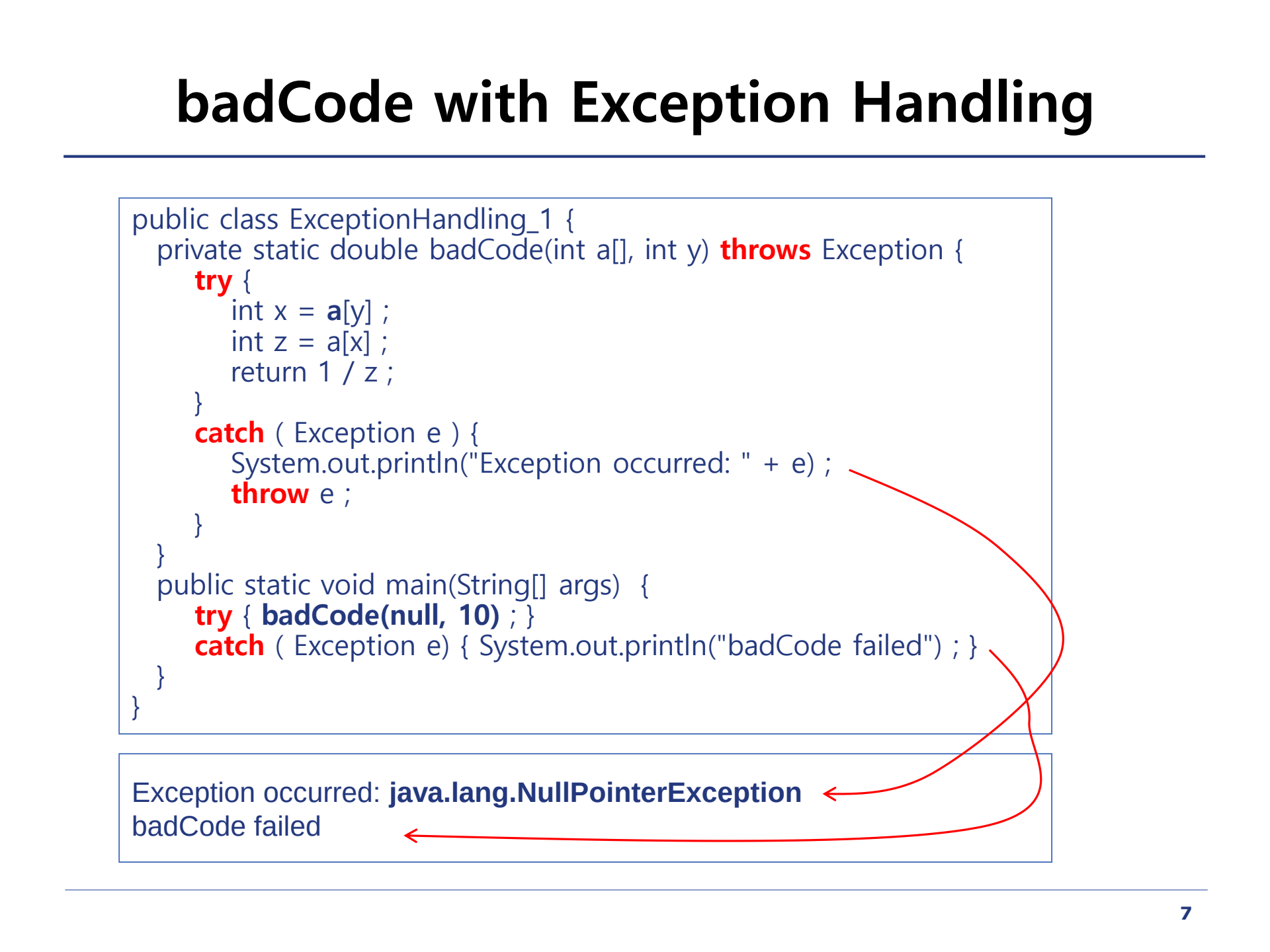
If an exception (error) occurs anywhere in the code inside the try block, the catch block is executed immediately

After the catch block (the catch handler) has finished, execution continues after the catch block

badCode with Exception Handling

```
public class ExceptionHandling_1 {
   private static double badCode(int a[], int y) throws Exception {
        try {
            int x = a[y] ;
            int z = a[x] ;
            return 1 / z ;
        }
        catch ( Exception e ) {
            System.out.println("Exception occurred: " + e) ;
            throw e ;
        }
   }
   public static void main(String[] args)  {
        try { badCode(null, 10) ; }
        catch ( Exception e) { System.out.println("badCode failed") ; }
   }
}
```

```
Exception occurred: java.lang.NullPointerException
badCode failed
```



badCode with Exception Handling

```
public class ExceptionHandling_2 {
   private static double badCode(int a[], int y) throws Exception {
        try {
            int x = a[y] ;
            int z = a[x] ;
            return 1 / z ;
        }
        catch ( Exception e ) {
            System.out.println("Exception occurred: " + e) ;
            throw e ;
        }
   }
   public static void main(String[] args)  {
        try { int[] a = {0, 1, 2} ; badCode(a, 3); }
        catch ( Exception e) { System.out.println("badCode failed") ; }
   }
}
```

```
Exception occurred: java.lang.ArrayIndexOutOfBoundsException:
                    Index 3 out of bounds for length 3
badCode failed
```

Standard Exceptions in Java

- Many useful exceptions are defined as subclasses of *RuntimeException* class in Java.
- They belong to java.lang package, not requiring importing

Exception	Description
NullPointerException	Thrown when an application attempts to use null in a case where an object is required
ArithmeticException	Thrown when an exceptional arithmetic condition has occurred; e.g.) divide by zero
ClassCastException	Thrown when an application attempts an illegal cast; for example Object x = new Integer(0); System.out.println((String)x);
ArrayIndexOutOfBoundsException	Thrown to indicate that an array has been accessed with an illegal index
NegativeArraySizeException	Thrown if an application tries to create an array with negative size.

Trace with Exception: printStackTrace()

```
01: public class ExceptionHandling_3 {
02:   private static double badCode(int a[], int y) throws Exception {
03:       try {
04:           int x = a[y] ; // throws ArrayIndexOutOfBoundsException
05:           int z = a[x] ;
06:           return 1 / z ;
07:       }
08:       catch ( Exception e ) {
09:           System.out.println("Exception occurred: " + e) ;
10:           e.printStackTrace();
11:           throw e ;
12:       }
13:   }
14:   public static void main(String[] args)  {
15:        try { int[] a = {0, 1, 2} ; badCode(a, 3); }
16:       catch ( Exception e) { System.out.println("badCode failed") ; }
17:   }
18: }
```

You can trace the source of the exception by printStackTrace()

```
Exception occurred: java.lang.ArrayIndexOutOfBoundsException: 3
java.lang.ArrayIndexOutOfBoundsException: 3
            at ExceptionHandling_1.badCode(ExceptionHandling_1.java:4)
            at ExceptionHandling_1.main(ExceptionHandling_1.java:15)
badCode failed
```

Multiple Catch Handlers

```
import java.util.Scanner;
public class MultipleCatchHandlers {
 public static void main(String[] args) {
    try {
       Scanner scanner = new Scanner(System.in) ;
       int x = scanner.nextInt() ; // can throw java.util.InputMismatchException
       int[] a = {-1, 0, 1, 2} ;
       int y = a[x] ; // can throw ArrayIndexOutOfBoundsException
       int z = a[y] / y ; // can throw ArrayIndexOutOfBoundsException and ArithmeticException
    }
    catch (ArithmeticException e) {
       System.out.println("Arithmetic exception took place: " + e) ;
    }
    catch (ArrayIndexOutOfBoundsException e) {
       System.out.println("Array index is invalid: " + e) ;
    }
    catch (java.util.InputMismatchException e) {
       System.out.println("The given string cannot be converted into an integer: " + e) ;
    }
 }
}
```

Multiple catch blocks can be provided to handle different types of exceptions.

```
    ...
    try {
       Scanner scanner = new Scanner(System.in) ;
       int x = scanner.nextInt() ;              // (3) java.util.InputMismatchException
       int[] a = {-1, 0, 1, 2} ;
       int y = a[x] ;                           // (1) ArrayIndexOutOfBoundsException
       int z = a[y] / y ;                       // (2) ArithmeticException
    }
    catch (ArithmeticException e) {
       System.out.println("Arithmetic exception took place: " + e) ;
    }
    catch (ArrayIndexOutOfBoundsException e) {
       System.out.println("Array index is invalid: " + e) ;
    }
    catch (java.util.InputMismatchException e) {
       System.out.println("The given string cannot be converted into an integer: " + e) ;
    }
 }
}
```

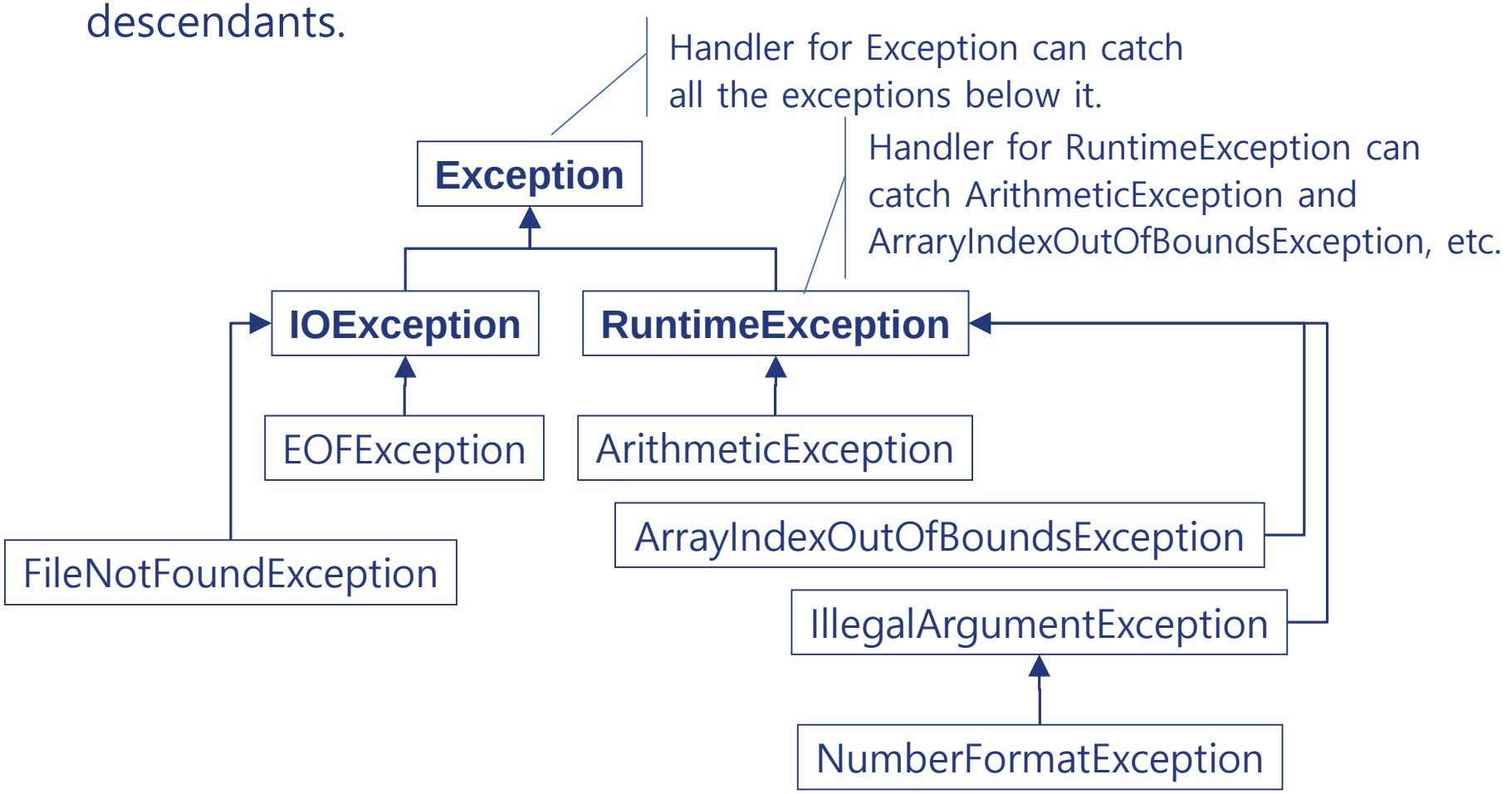
0
Array index is invalid: **java.lang.ArrayIndexOutOfBoundsException: -1**

1
Arithmetic exception took place: **java.lang.ArithmeticException: / by zero**

abc
The given string cannot be converted into an integer: **java.util.InputMismatchException**

Catch Handler for Multiple Exceptions

- Catch handler can catch all the exceptions of its own type and its descendants.



```
import java.io.*;
public class CatchingMultipleExceptions {
 public static void main(String[] args) {
    try {
       FileReader fin = new FileReader("employee.txt") ; // can throw FileNotFoundException
       BufferedReader in = new BufferedReader(fin) ;

       int i1 = Integer.parseInt(in.readLine()) ; // can throw NumberFormatException
       int i2 = Integer.parseInt(in.readLine()) ; // can throw NumberFormatException
       System.out.printf("%d %d%n ", i1, i2) ;

       int[] a = {-10, 0, 10, 20} ;
       System.out.println(a[i1] / a[i2]) ;
       // can throw ArrayIndexOutOfBoundsException and ArithmeticException
    }
    catch ( IOException e) { // can catch java.io.FileNotFoundException
       System.out.println(e) ;
    }
    catch (RuntimeException e) {
       // can catch ArrayIndexOutOfBoundsException, ArithmeticException, and NumberFormatException
       System.out.println(e) ;
    }
 }
}
```

```
2 1
 java.lang.ArithmeticException: / by zero
```

```
0 4
 java.lang.ArrayIndexOutOfBoundsException: 4
```

The Most General Hander

```
catch (Exception e) {
      …
}
```

can catch any exception because **Exception** is the superclass of all exception types

```
import java.io.*;
public class CatchingMultipleExceptions {
 public static void main(String[] args) {
    try {
       …
    }
    catch (Exception e) { System.out.println(e) ; }
    catch ( IOException e) { System.out.println(e) ;  }
    catch (RuntimeException e) { System.out.println(e) ; }
 }
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
      Unreachable catch block for IOException.
      It is already handled by the catch block for Exception
      Unreachable catch block for RuntimeException.
      It is already handled by the catch block for Exception
```

Subclasses of Exception

java.lang
Class Exception

java.lang.Object
  └ java.lang.Throwable
    └ **java.lang.Exception**

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AclNotFoundException, ActivationException, AlreadyBoundException, ApplicationException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CertificateException, ClassNotFoundException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSException, IllegalAccessException, IllegalClassFormatException, InstantiationException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, InvocationTargetException, IOException, JMException, LastOwnerException, LineUnavailableException, MidiUnavailableException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NoSuchFieldException, NoSuchMethodException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, RefreshFailedException, RemarshalException, RuntimeException, SAXException, ServerNotActiveException, SQLException, TimeoutException, TooManyListenersException, TransformerException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URISyntaxException, UserException, XAException, XMLParseException, XPathException

Subclasses of IOException

java.io
Class IOException

java.lang.Object
  └ java.lang.Throwable
    └ java.lang.Exception
      └ **java.io.IOException**

All Implemented Interfaces:
Serializable

Direct Known Subclasses:
ChangedCharSetException, CharacterCodingException, CharConversionException, ClosedChannelException, EOFException, FileLockInterruptionException, FileNotFoundException, HttpRetryException, IIOException, InterruptedIOException, InvalidPropertiesFormatException, JMXProviderException, JMXServerErrorException, MalformedURLException, ObjectStreamException, ProtocolException, RemoteException, SaslException, SocketException, SSLException, SyncFailedException, UnknownHostException, UnknownServiceException, UnsupportedEncodingException, UTFDataFormatException, ZipException

Subclasses of RuntimeException

java.lang
Class RuntimeException

java.lang.Object
  └ java.lang.Throwable
    └ java.lang.Exception
      └ **java.lang.RuntimeException**

All Implemented Interfaces:
Serializable

Direct Known Subclasses:
AnnotationTypeMismatchException, ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, DOMException, EmptyStackException, EnumConstantNotPresentException, EventException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, ImagingOpException, IncompleteAnnotationException, IndexOutOfBoundsException, JMRuntimeException, LSException, MalformedParameterizedTypeException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NullPointerException, ProfileDataException, ProviderException, RasterFormatException, RejectedExecutionException, SecurityException, SystemException, TypeNotPresentException, UndeclaredThrowableException, UnmodifiableSetException, UnsupportedOperationException

The finally Block

- The finally block *always* executes

```
public class FinallyBlock {
    public static void main(String[] args) {
       String[] array = {"First", "Second", "Third"};
       List<String> list = new ArrayList<>(Arrays.asList(array));
       PrintWriter out = null;
       try {
           System.err.println("Entering try statement");
           out = new PrintWriter( new FileWriter("OutFile.txt"));
           for (int i = 0; i < 10; i++) out.println("Value at: " + i + " = " + list.get(i));
       } catch (IndexOutOfBoundsException e) {
           System.err.println("Caught IndexOutOfBoundsException: " + e.getMessage());
       } catch (IOException e) {
           System.err.println("Caught IOException: " + e.getMessage());
       } finally {
           if (out != null) { System.err.println("Closing PrintWriter"); out.close(); }
           else { System.err.println("PrintWriter not open"); }
       }
    }
}
```

```
Entering try statement
Caught IndexOutOfBoundsException: Index 3 out of bounds for length 3
Closing PrintWriter
```

Closing a File with Finally Block

- You can use a finally block to ensure that a resource is closed regardless of whether the try statement completes normally or abruptly.

```
public static String readFirstLine(String path) throws IOException {
      BufferedReader br = null;
      String line = "";
      try {
          br = new BufferedReader(new FileReader(path));
          line = br.readLine();
      } finally {
          if ( br != null ) br.close(); // may throw exception
      }
      return line;
   }
```

Closing a File with Finally Block: Problem

- Close() can fail and throw an exception. Thus, close() needs to be tried.

```
public static String readFirstLine(String path) throws IOException {
    BufferedReader br = null;
    String line = "";
    try {
        br = new BufferedReader(new FileReader(path));
        line = br.readLine();
    } finally {
        try {
            if ( br != null ) br.close();
        } catch (IOException e) {

        }
    }
    return line;
}
```

The try-with-resources Statement

- The try-with-resources statement ensures that each resource is closed at the end of the statement.
- A resource: any object that implements java.lang.AutoCloseable can be used as a resource.

```
public static String readFirstLine(String path) throws IOException {
    String line = "";
    try ( BufferedReader br = new BufferedReader(new FileReader(path)) )
    {
      line = br.readLine();
    }
    return line;
}
```

User-defined Exceptions: definitions

- Programmer can define new classes of exceptions by extending, commonly, the RuntimeException class.

```
class MyDivideByZeroException extends RuntimeException {
    public String toString() {
        return "Zero denominator used" ;
    }
}

class MyArrayOutOfBoundsException extends RuntimeException {
    private int invalidIndex ;

    public MyArrayOutOfBoundsException(int invalidIndex) {
        this.invalidIndex = invalidIndex ;
    }
    public String toString() {
        return String.format("Invalid Array Index: %d", invalidIndex) ;
    }
}
```

User-defined Exceptions: Throwing and Catching

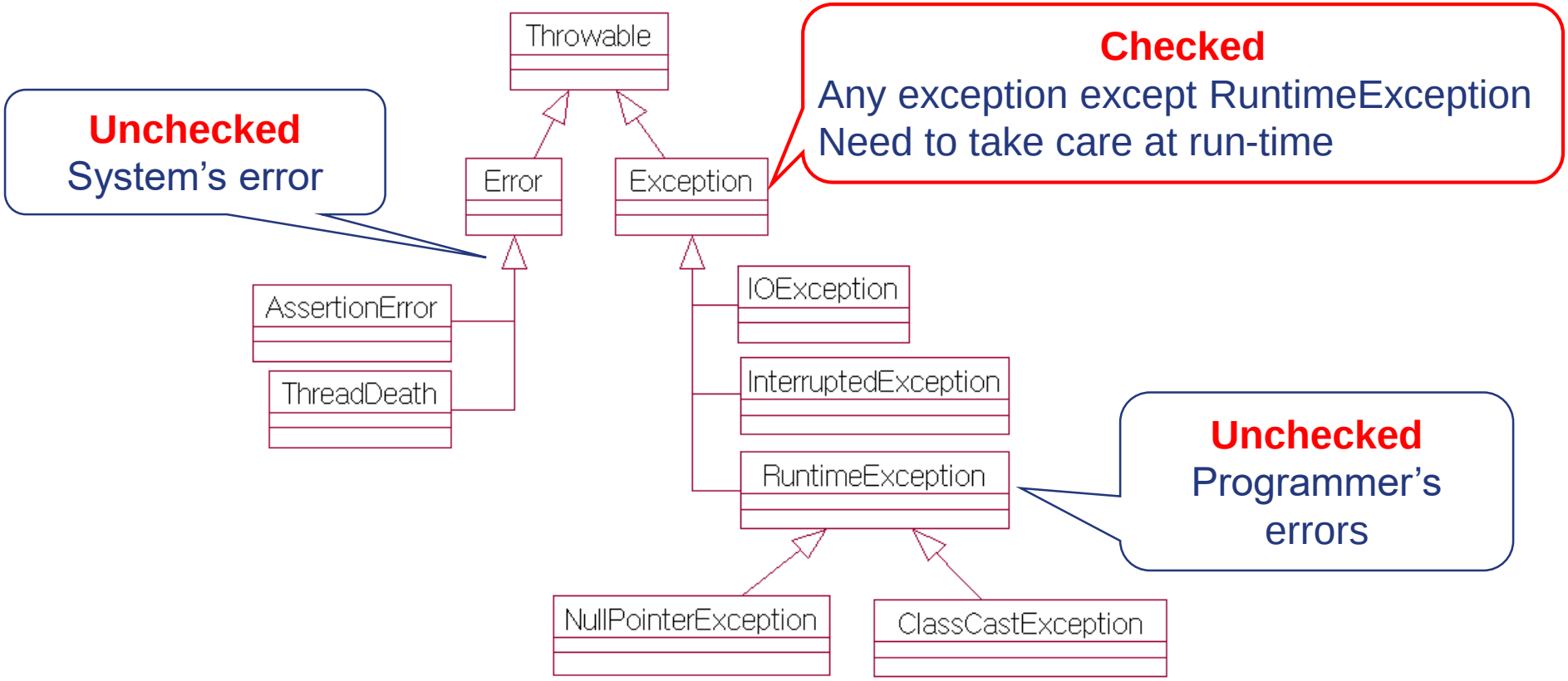
```
public class UserDefinedException {
  private static double badCode(int a[], int y) {
    if ( y < 0 || y >= a.length ) throw new MyArrayOutOfBoundsException(y) ;
    int x = a[y] ;

    if ( x == 0 ) throw new MyDivideByZeroException() ;
    return 1 / x ;
  }

  public static void main(String[] args) {
    try {
        int[] a = {0, 1, 2} ;
        badCode(a, 5) ; // throw MyArrayOutOfBoundsException(y)
        // badCode(a, 0) ; // throw MyDivideByZeroException()
    }
    catch (MyDivideByZeroException e ) { System.out.println(e) ; }
    catch ( MyArrayOutOfBoundsException e) { System.out.println(e) ; }
  }
}
```

Types of Exceptions

- Exceptions are classified into **checked** and **unchecked**



How to handle checked exceptions

- For checked exceptions, you should catch them

```
class MyRangeException extends Exception {
   public String toString() { return “MyRangeException" ; }
}
class MyNumberException extends RuntimeException {
   public String toString() { return " MyNumberException " ; }
}
public class CheckedException {
   public static void write() {
       try {
           FileReader fin = new FileReader("employee.txt") ;
           throw new MyNumberException() ;
           throw new MyRangeException() ;
       }
       catch ( IOException e) { … }
       catch ( MyRangeException e) { … }
   }
}
```

How to handle checked exceptions

- Or, you should rethrow, so that the caller can handle them

```
class MyRangeException extends Exception {
   public String toString() { return “MyRangeException" ; }
}
class MyNumberException extends RuntimeException {
   public String toString() { return " MyNumberException " ; }
}
public class CheckedException {
   private static void write() throws IOException, MyRangeException {
            FileReader fin = new FileReader("employee.txt") ;
            throw new MyNumberException() ;
            throw new MyRangeException() ;
   }
   public static void main(String[] args) throws IOException, MyRangeException {
       write() ;
   }
}
```

How to handle checked exceptions

- If there are no catch or rethrows for checked exceptions, compiler will issue an error message

```
public class CheckedException {
   private static void write() {
          FileReader fin = new FileReader("employee.txt") ;
          // unhandled exception type FileNotFoundException
          throw new MyNumberException() ;
          throw new MyRangeException() ;
          // unhandled exception type MyRangeException
   }
}
```

```
public class CheckedException {
   private static void write() throws IOException, MyRangeException {
          FileReader fin = new FileReader("employee.txt") ;
          throw new MyNumberException() ;
          throw new MyRangeException() ;
   }
   public static void main(String[] args)  {
       write() ;
   }
}
```

Not Handling an Exception

- If a method raises an exception and it does *not* have a catch block or throws declaration then…
  - ***checked exception***: the compiler will reject your code at compile time: You should catch or rethrows it !
  - ***unchecked exception:*** the program will terminate at runtime

Not Handling an Unchecked Exception

- Unhandled unchecked exceptions

```
1: class MyNumberException extends RuntimeException {
2:      public String toString() { return "MyNumberException" ; }
3: }
4: public class UncheckedException {
5:      public static void main(String[] args) {
6:          write() ; // not handle MyNumberException
7:      }
8:      public static void write()  {
9:          throw new MyNumberException() ;
10:     }
   }
```

```
Exception in thread "main" MyNumberException
            at UncheckedException.write(UncheckedException.java:9)
            at UncheckedException.main(UncheckedException.java:6)
```

Stack Trace Information

- If no handler is called, then the system prints a *stack trace* as the program terminates
  - it is a list of the called methods that are waiting to return when the exception occurred
  - *very useful* for debugging/testing
- The stack trace can also be printed by calling `printStackTrace()`

```
01: public class UsingStackTrace {
02:  public static void main( String args[] ) {
03:       try { method1(); /* call method1*/ }
04:      catch ( Exception exception ) {
05:          System.err.println( exception.getMessage() + "₩n" );
06:          exception.printStackTrace();
07:          StackTraceElement[] traceElements = exception.getStackTrace();
08:          System.out.println(  "₩nStack trace from getStackTrace:" );
09:          System.out.println( "Class₩t₩tFile₩t₩t₩tLine₩tMethod" );
10:          for ( int i = 0; i < traceElements.length; i++ ) {
11:             StackTraceElement element = traceElements[ i ];
12:             System.out.print( element.getClassName() + "₩t" );
13:             System.out.print( element.getFileName() + "₩t" );
14:             System.out.print( element.getLineNumber() + "₩t" );
15:             System.out.print( element.getMethodName() + "₩n" );
16:          } // end for block
17:       } // end catch
18:    } // end method main
19:    public static void method1() throws Exception { method2(); }
20:    public static void method2() throws Exception { method3(); }
21:    public static void method3() throws Exception { throw new Exception( "Exception
 thrown in method3" ); }
} // end class Using Exceptions
```

The compiler will reject the program at compile time if the throws are not included because Exception is checked exception

```
Exception thrown in method3

java.lang.Exception: Exception thrown in method3
        at UsingStackTrace.method3(UsingStackTrace.java:21)
        at UsingStackTrace.method2(UsingStackTrace.java:20)
        at UsingStackTrace.method1(UsingStackTrace.java:19)
        at UsingStackTrace.main(UsingStackTrace.java:3)

Stack trace from getStackTrace:
Class                File                    Line    Method
UsingStackTrace UsingStackTrace.java    21      method3
UsingStackTrace UsingStackTrace.java    20      method2
UsingStackTrace UsingStackTrace.java    19      method1
UsingStackTrace UsingStackTrace.java    3       main
```

Using Assertions

- With assertions, you can use assertions to check some conditions.

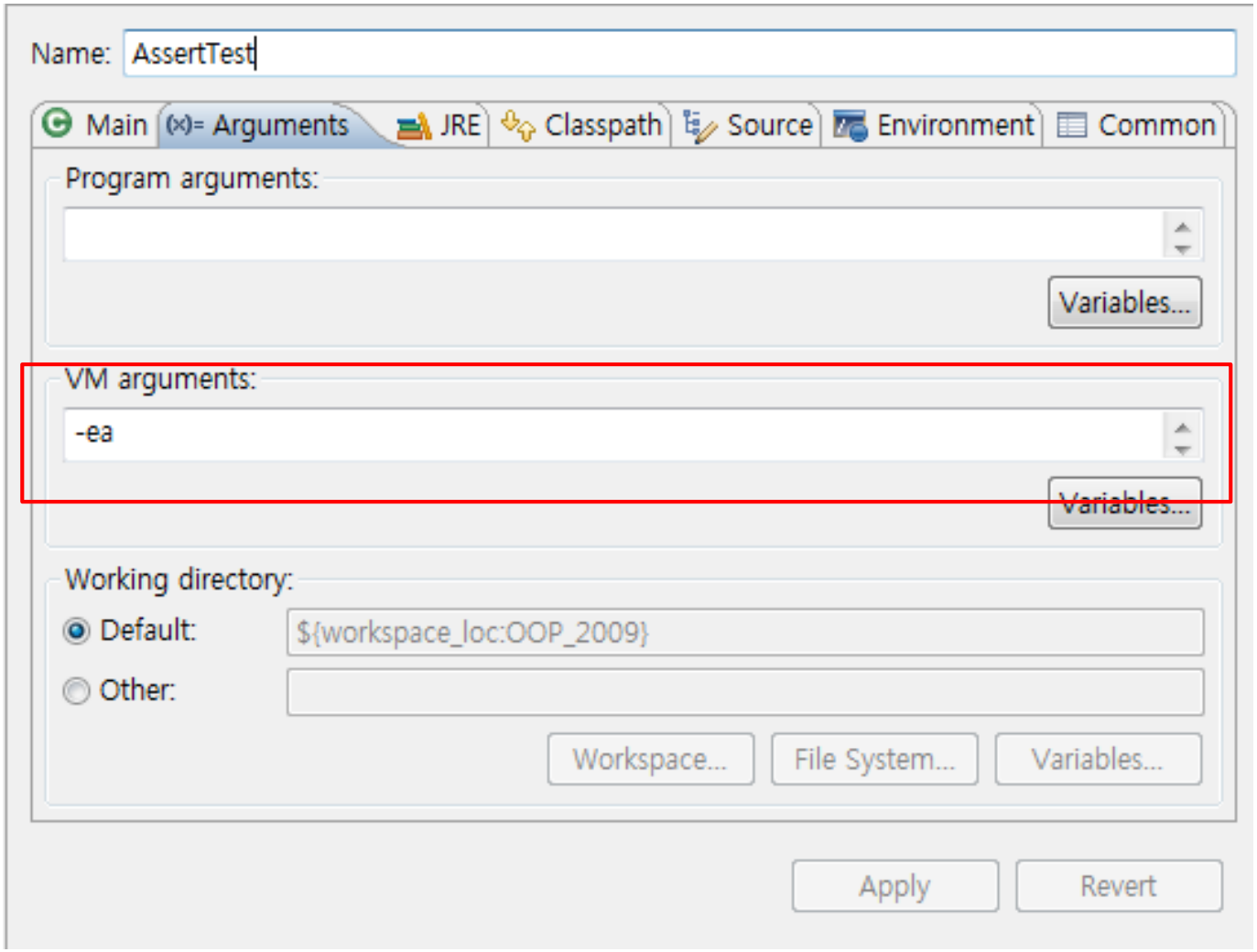
```
1: public class AssertTest {
2:   public static void main(String args[]) {
3:       assert args.length > 0 : "₩nusage: java AssertTest <args>" ;
4:       int i = 0;
5:       do {
6:           System.out.println(args[i]);
7:           i++;
8:       } while (i < args.length);
9:   }
  }
```

```
Exception in thread "main" java.lang.AssertionError:
usage: java AssertTest <args>
        at AssertTest.main(AssertTest.java:3)
```

Using Assertions

- Usages for assert
  - assert condition ;
  - Assert condition : expression ;
- Assertions can be enabled and disabled by –ea option
  - java –ea AssertTest
  - java –da AssertTest

Enabling and Disabling assertions in Eclipse



Enabling and Disabling assertions: Comparison

- When assertions are enabled; java –ea AssertTest

```
Exception in thread "main" java.lang.AssertionError:
usage: java AssertTest <args>
        at AssertTest.main(AssertTest.java:3)
```

- When assertions are disabled; java –da AssertTest

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
        Index 0 out of bounds for length 0
        at AssertTest.main(AssertTest.java:6)
```

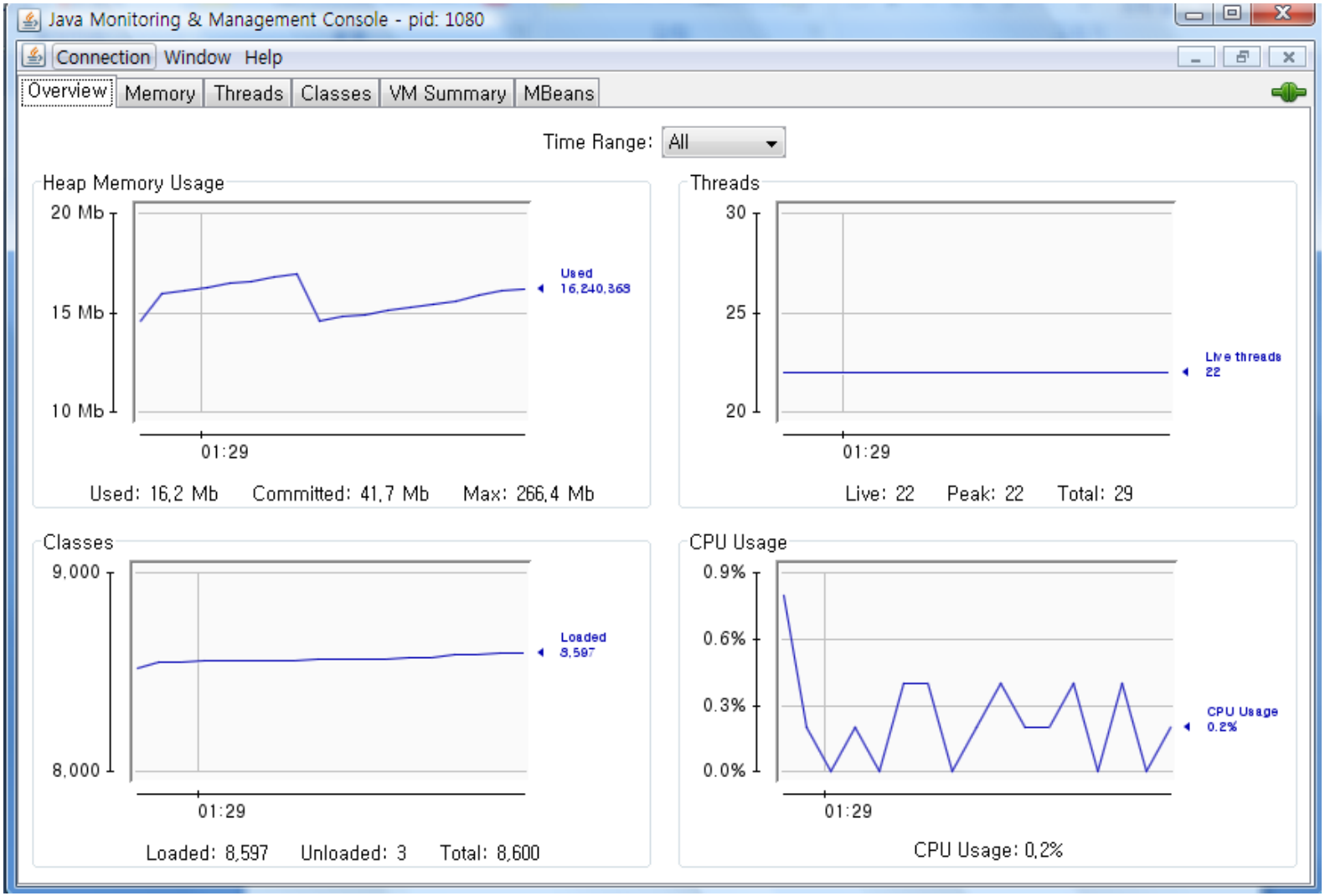
Checking Pre/post condition by assertions

- Precondition: conditions that should be satisfied before the execution
- Postcondition: conditions that should be satisfied after the execution

```
public class PrePostCondition {
    public static void sort(int []a) {
        assert a != null ;                          // precondition
        for ( int i = 0 ; i < a.length - 1; i ++ )
            for ( int j = i + 1 ; j < a.length - 1; j ++ )
                if ( a[i] < a[j] ) {
                        int t = a[i] ; a[i] = a[j] ; a[j] = t ;
                }
        for ( int i = 0 ; i < a.length - 1; i ++ )    // postcondition
            assert a[i] >= a[i+1] ;
    }
    public static void main(String[] args) {
        int[] a = {0, 25, 20, 50} ;
        sort(a) ;
    }
}
```

```
Exception in thread "main" java.lang.AssertionError
        at PrePostCondition.sort(PrePostCondition.java:12)
        at PrePostCondition.main(PrePostCondition.java:16)
```

jconsole: Monitoring JVM



Profiling Application

```
public class ProfilerTest {
    public static void sort(int []a) {
        assert a != null ;
        for ( int i = 0 ; i < a.length - 1; i ++ )
           for ( int j = i + 1 ; j < a.length ; j ++ )
               if ( a[i] < a[j] ) { int t = a[i] ; a[i] = a[j] ; a[j] = t ; }
        for ( int i = 0 ; i < a.length - 1; i ++ )
           assert a[i] >= a[i+1] ;
    }
    public static void read(int []a) { for ( int i = 0 ; i < a.length ; i ++ ) a[i] = i ; }
    public static void print(int []a) {
        for ( int i = 0 ; i < a.length ; i ++ ) System.out.println(a[i]) ;
    }
    public static void main(String[] args) {
        final int SIZE = 10 * 10 * 10 * 10 * 10 ; // 100K
        int[] a = new int[SIZE] ;
        read(a) ;
        sort(a) ;
        print(a) ;
    }
}
```

Profiling Application

- Run the app with –Xprof option

```
Flat profile of 28.19 secs (1797 total ticks): main

  Interpreted + native   Method
  0.1%     0  +     2    java.io.FileOutputStream.writeBytes
  0.1%     0  +     2    Total interpreted

     Compiled + native   Method
 94.4%  1697  +     0    ProfilerTest.sort
  0.1%     1  +     0    sun.nio.cs.ext.DoubleByteEncoder.encodeArrayLoop
 94.5%  1698  +     0    Total compiled

        Stub + native   Method
  5.4%     0  +    97    java.io.FileOutputStream.writeBytes
  5.4%     0  +    97    Total stub

  …
```