What's are Loaded / Reloaded to Java 7

Krishantha Dinesh
[kdinesh@virtusa.com]

@2013 Oct

2000 West Park Drive
Westborough MA 01581 USA
Phone: 508 389 7300 Fax: 508 366 9901

virtusa®
Accelerating Business Outcomes

Objectives

- Not to teach Java
- Learn about what has been changed on JDK 1.7 and what's are new

Prerequisites

- Java.basicknowladge >(60/100);
- Java.version6.complete=true;
- Mobilephone.soundmode=soundmode.completesilent && soundmode.completesilent != soundmode.vibrate

Release History

Version	Released Date
JDK Alpha and Beta	1995
JDK 1.0	January 23, 1996
JDK 1.1	February 19, 1997
J2SE 1.2	December 8, 1998
J2SE 1.3	May 8, 2000
J2SE 1.4	February 6, 2002
J2SE 5.0	September 30, 2004
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8	Java 8 is expected in March 2014
Java SE 9	At JavaOne 2011, Oracle discussed features they hope to have in a 2016 release of Java 9, including better support for multi-gigabyte heaps, better native code integration, and a self-tuning JVM
Java SE 10	There is speculation of removing primitive data types and move towards 64-bit addressable arrays to support large data sets

virtusa®

What is Project COIN

- Project Coin, a central part of Java 7, was described by Darcy [Joseph Darcy, Member of the Oracle Technical Staff] as “a suite of language and library changes to make things programmers do everyday easier.”
- Project Coin has strong IDE support:
  - • IntelliJ IDEA 10.5 and later
  - • Eclipse 3.7.1 and later
  - • NetBeans 7.0 and later

Project COIN inside

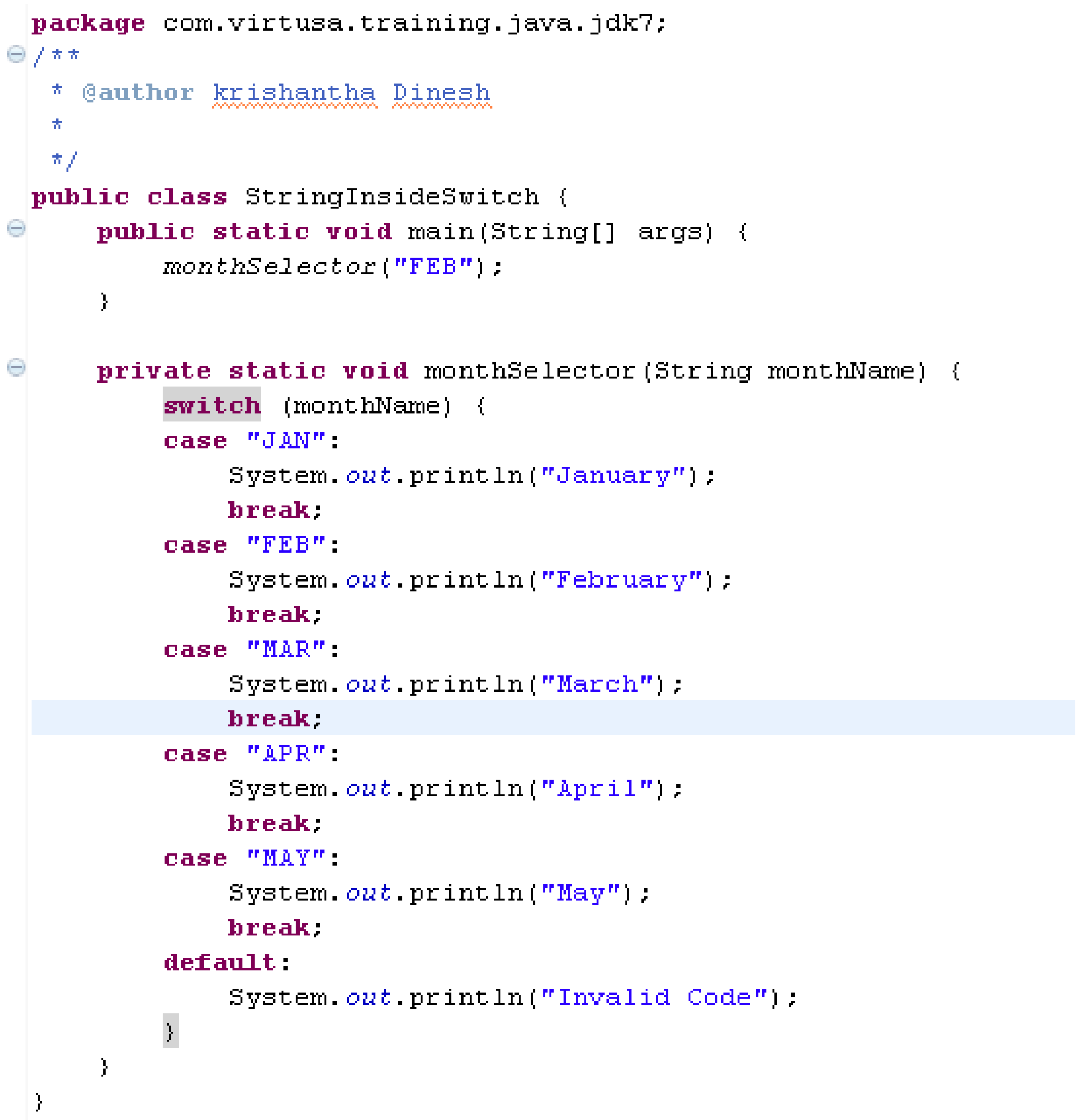
The six Project Coin features are:

- Strings in switch
- Binary literals and underscores in literals
- Diamond
- Multi-catch and more precise re-throw
- try-with-resources
- Varargs warnings

String in Switch

- Before JDK 7 only byte, short, char, int were allowed in switch statements
- JDK 7 allows Strings to be used in switch statements

String in Switch - example



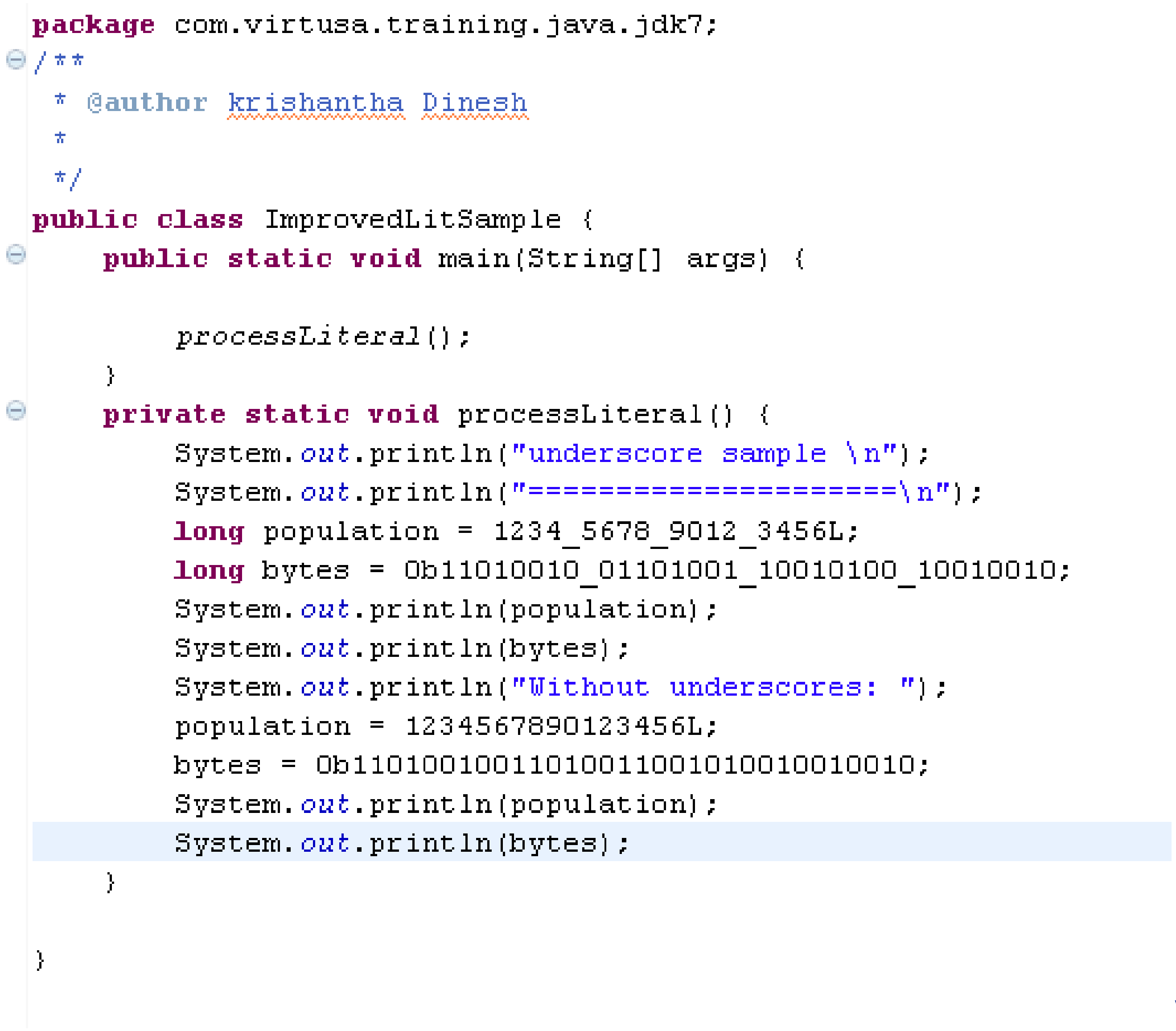
```
package com.virtusa.training.java.jdk7;
/**
 * @author krishantha Dinesh
 *
 */
public class StringInsideSwitch {
    public static void main(String[] args) {
        monthSelector("FEB");
    }

    private static void monthSelector(String monthName) {
        switch (monthName) {
        case "JAN":
            System.out.println("January");
            break;
        case "FEB":
            System.out.println("February");
            break;
        case "MAR":
            System.out.println("March");
            break;
        case "APR":
            System.out.println("April");
            break;
        case "MAY":
            System.out.println("May");
            break;
        default:
            System.out.println("Invalid Code");
        }
    }
}
```

Improved literals

- A literal is the source code representation of a fixed value.
- In Java SE 7 and later, any number of underscore characters (_) can appear anywhere between digits in a numerical literal. This feature enables you to separate groups of digits in numeric literals, which can improve the readability of your code.

```
long population= 1234_5678_9012_3456L;

long SecurityNumber = 999_99_9999L;

float pi =  3.14_15F;

long hexBytes = 0xFF_EC_DE_5E;

long hexWords = 0xCAFE_BABE;

long maxLong = 0x7fff_ffff_ffff_ffffL;

byte nybbles = 0b0010_0101;

long bytes = 0b11010010_01101001_10010100_10010010;
```

Improved literals - Example



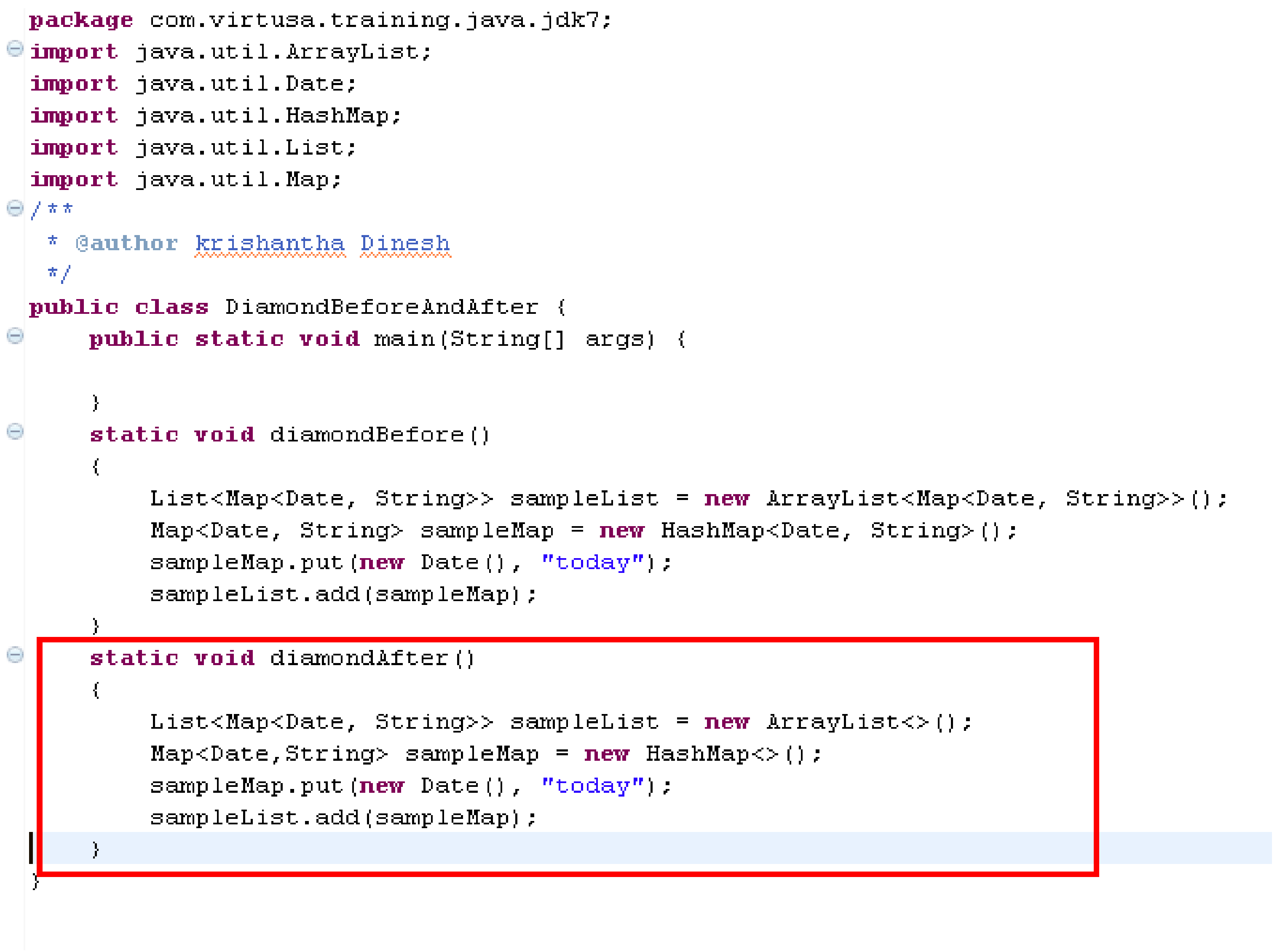
```java
package com.virtusa.training.java.jdk7;
/**
 * @author krishantha Dinesh
 *
 */
public class ImprovedLitSample {
    public static void main(String[] args) {

        processLiteral();
    }
    private static void processLiteral() {
        System.out.println("underscore sample \n");
        System.out.println("=====================\n");
        long population = 1234_5678_9012_3456L;
        long bytes = 0b11010010_01101001_10010100_10010010;
        System.out.println(population);
        System.out.println(bytes);
        System.out.println("Without underscores: ");
        population = 1234567890123456L;
        bytes = 0b11010010011010011001010010010010;
        System.out.println(population);
        System.out.println(bytes);
    }

}
```

Diamond Operator (<>)

As you can see in the right part of the assignment in lines you need to repeat your type information for the sampleList variable as well as of the sampleMap variable.

```
static void diamondBefore()
{

    List<Map<Date, String>> sampleList = new ArrayList<Map<Date, String>>();
    Map<Date, String> sampleMap = new HashMap<Date, String>();
    sampleMap.put(new Date(), "today");
    sampleList.add(sampleMap);
}
```

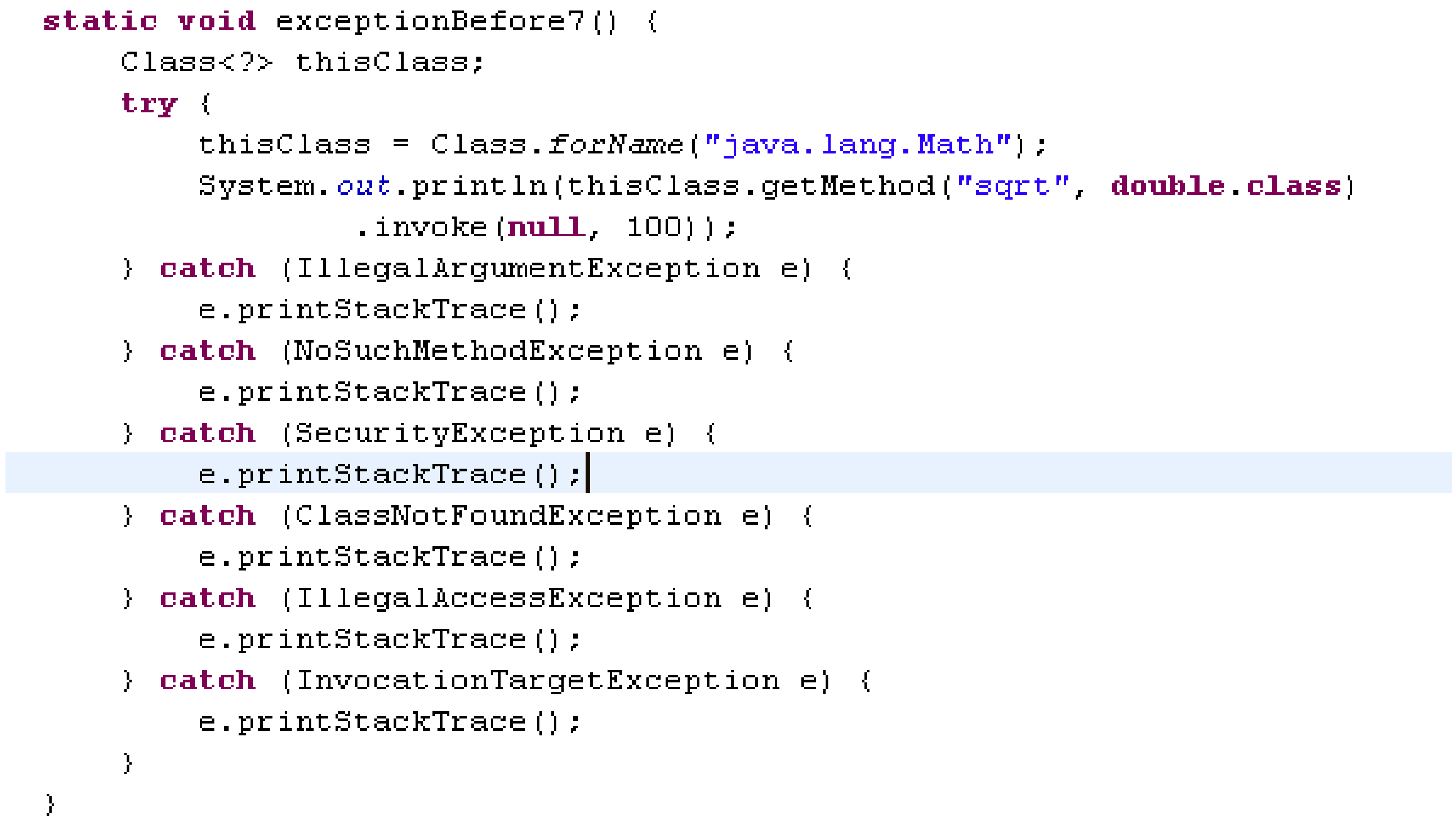
Diamond Operator in java 7



```
package com.virtusa.training.java.jdk7;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
/**
 * @author krishantha Dinesh
 */
public class DiamondBeforeAndAfter {
    public static void main(String[] args) {

    }
    static void diamondBefore()
    {
        List<Map<Date, String>> sampleList = new ArrayList<Map<Date, String>>();
        Map<Date, String> sampleMap = new HashMap<Date, String>();
        sampleMap.put(new Date(), "today");
        sampleList.add(sampleMap);
    }
    static void diamondAfter()
    {
        List<Map<Date, String>> sampleList = new ArrayList<>();
        Map<Date,String> sampleMap = new HashMap<>();
        sampleMap.put(new Date(), "today");
        sampleList.add(sampleMap);
    }
}
```

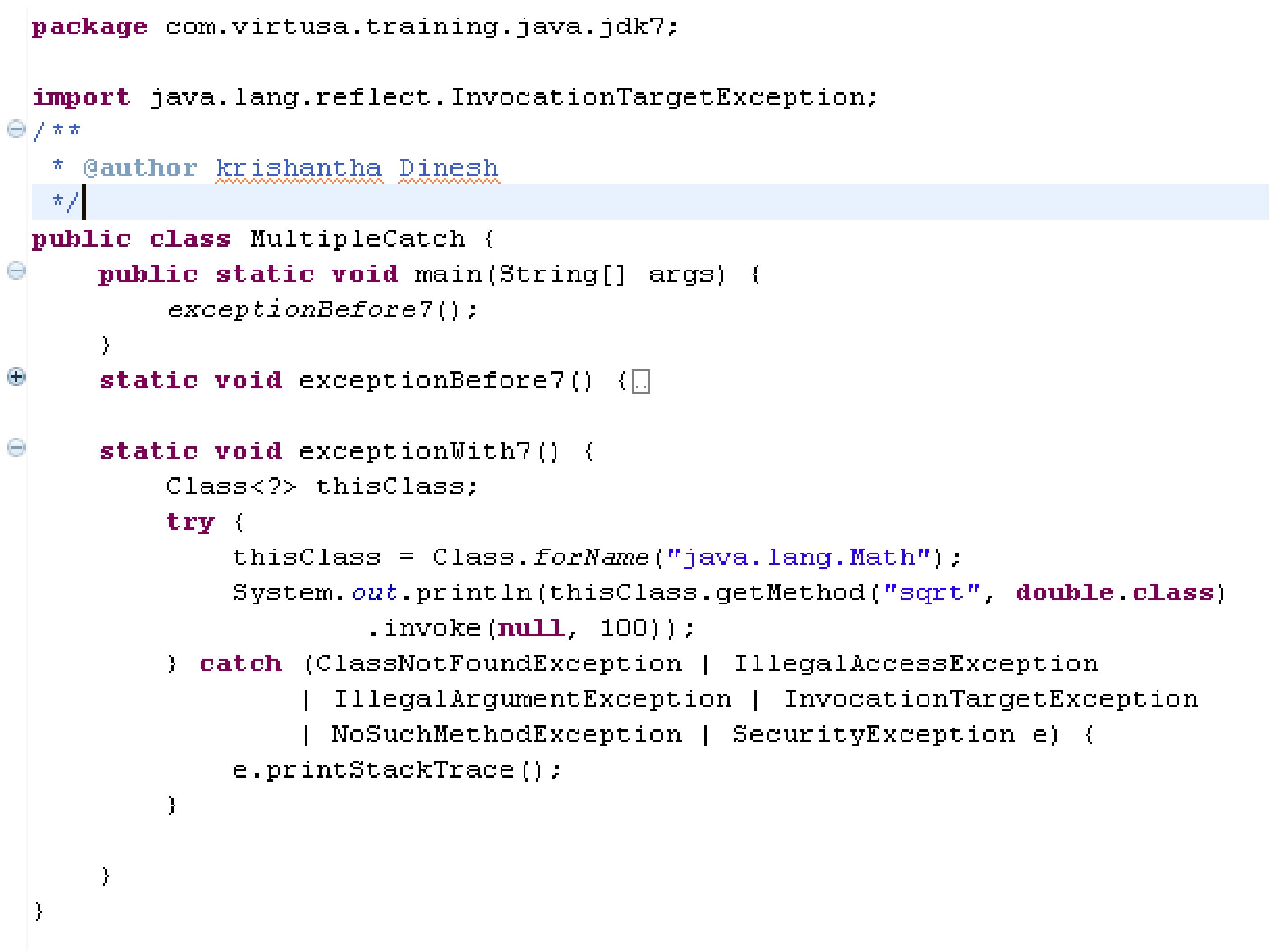
Exception Handling – before java 7

- Before Java 7 exception handle was like this



```java
static void exceptionBefore7() {
    Class<?> thisClass;
    try {
        thisClass = Class.forName("java.lang.Math");
        System.out.println(thisClass.getMethod("sqrt", double.class)
                .invoke(null, 100));
    } catch (IllegalArgumentException e) {
        e.printStackTrace();
    } catch (NoSuchMethodException e) {
        e.printStackTrace();
    } catch (SecurityException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    }
}
```

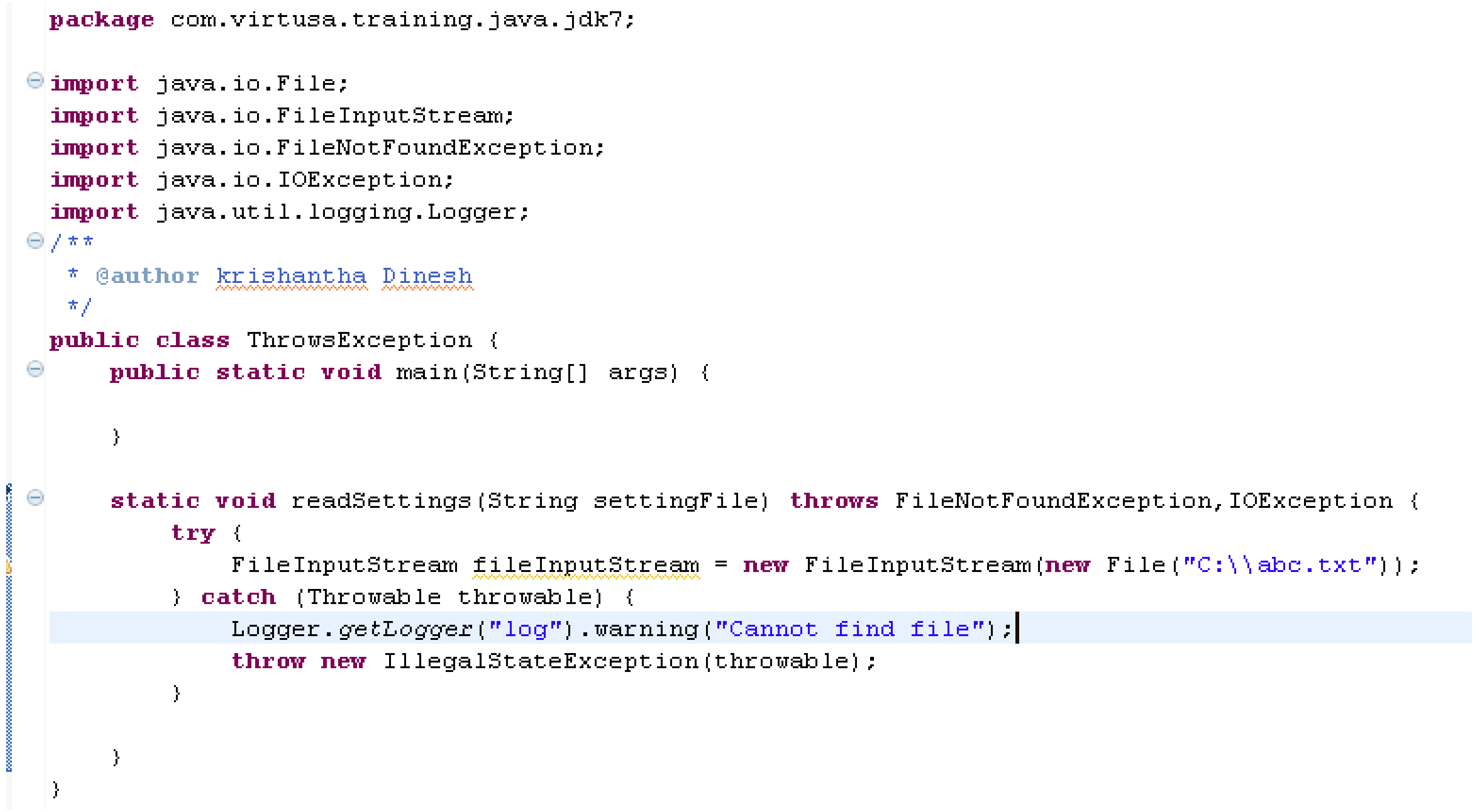
With java 7 we have multiple catch



```
package com.virtusa.training.java.jdk7;

import java.lang.reflect.InvocationTargetException;
/**
 * @author krishantha Dinesh
 */
public class MultipleCatch {
    public static void main(String[] args) {
        exceptionBefore7();
    }
    static void exceptionBefore7() {

    static void exceptionWith7() {
        Class<?> thisClass;
        try {
            thisClass = Class.forName("java.lang.Math");
            System.out.println(thisClass.getMethod("sqrt", double.class)
                    .invoke(null, 100));
        } catch (ClassNotFoundException | IllegalAccessException
                | IllegalArgumentException | InvocationTargetException
                | NoSuchMethodException | SecurityException e) {
            e.printStackTrace();
        }

    }
}
```

Single re-throw

- Why need re-throw
  - If we need to do something before throws like logging
  - If we need to convert exception to other type of exception



```java
package com.virtusa.training.java.jdk7;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.logging.Logger;
/**
 * @author krishantha Dinesh
 */
public class ThrowsException {
    public static void main(String[] args) {

    }

    static void readSettings(String settingFile) throws FileNotFoundException,IOException {
        try {
            FileInputStream fileInputStream = new FileInputStream(new File("C:\\abc.txt"));
        } catch (Throwable throwable) {
            Logger.getLogger("log").warning("Cannot find file");
            throw new IllegalStateException(throwable);
        }

    }
}
```

Try with Resources

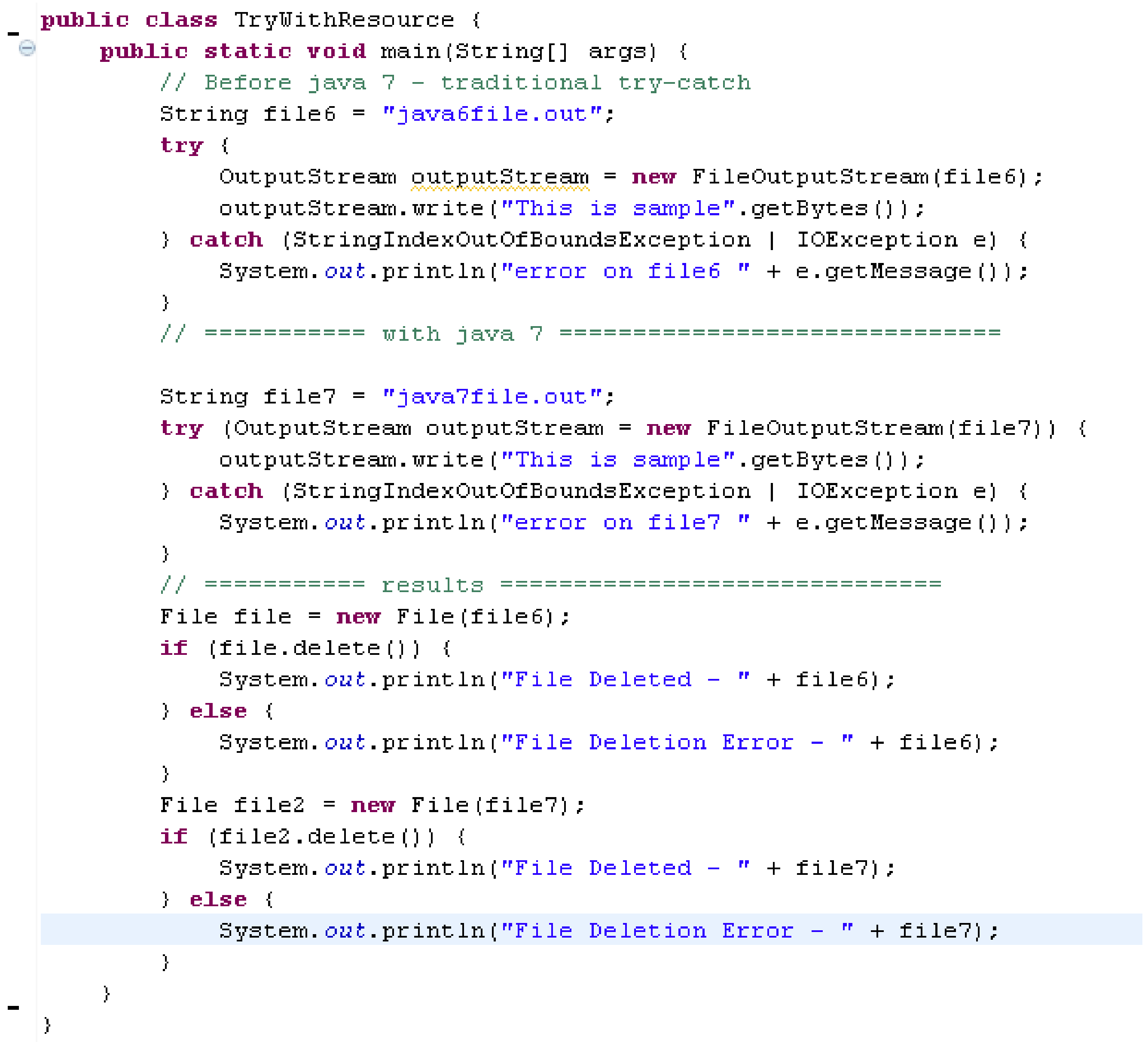
- This feature really helps in terms of reducing unexpected runtime exceptions for your code
- In Java 7 you can use the try-with-resource clause that automatically closes all open resources if an exception occurs.
- Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource
  - Before 7

```
try{ //open file or resources }
catch(IOException){ //handle exception }
finally{ //close file or resources }
```

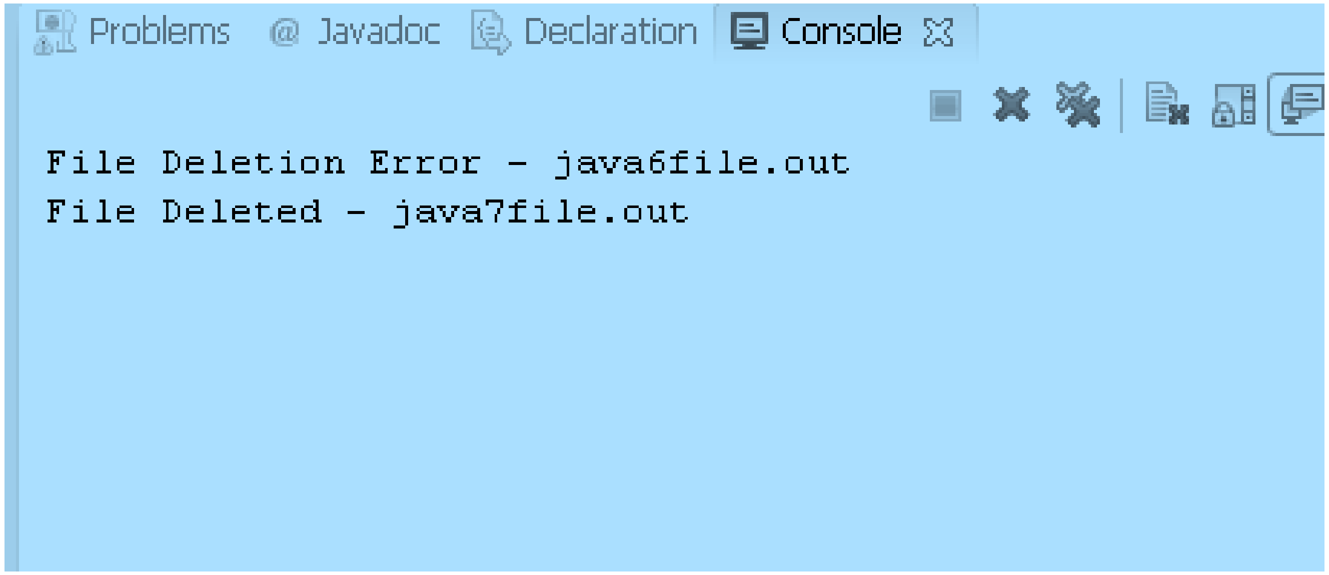
  - With 7

-
  ```
  try(open file or resource here)
  { //… }
   //after try block, file will close automatically.
  ```

Try with Resources Example

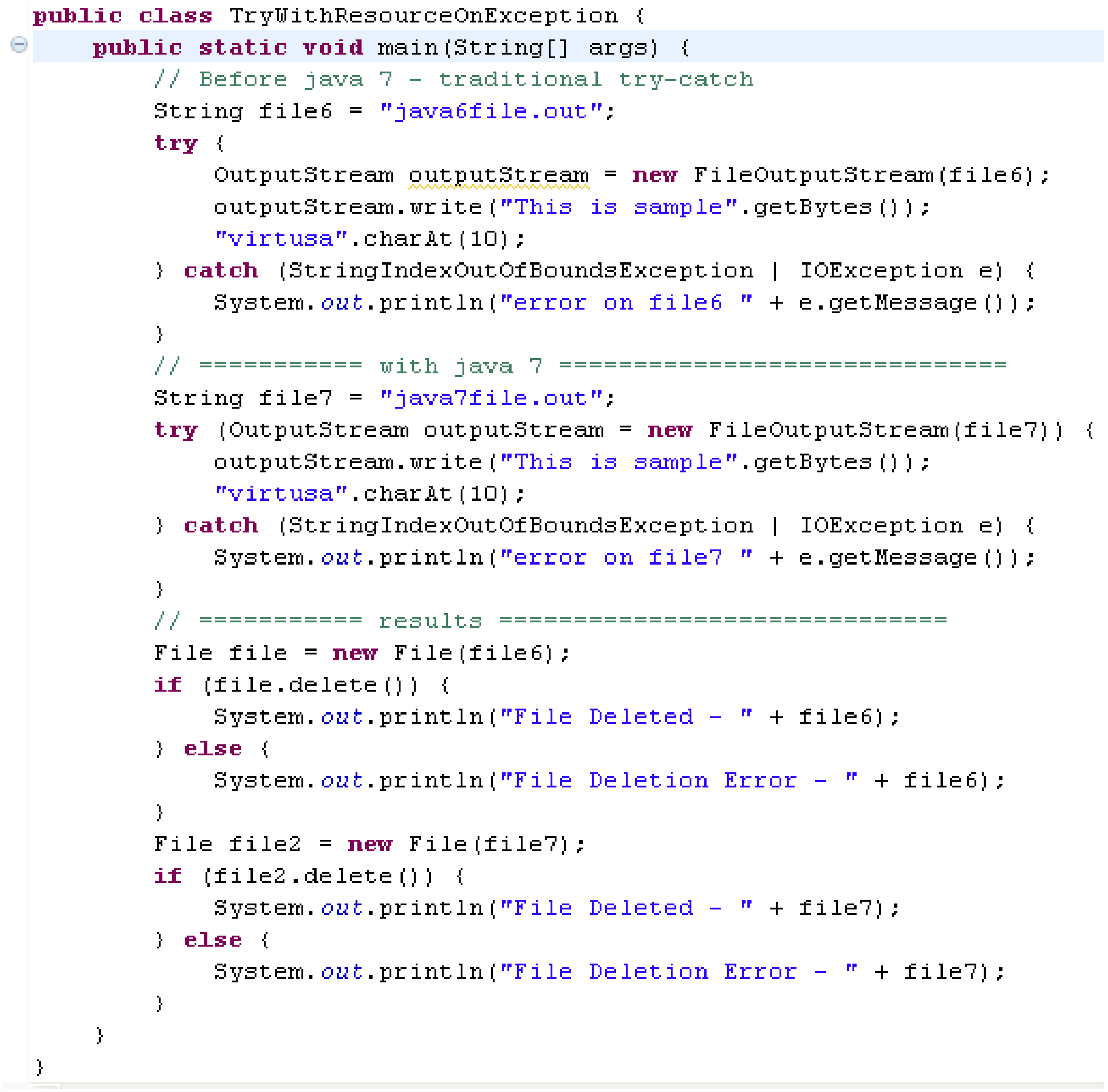


```
public class TryWithResource {
    public static void main(String[] args) {
        // Before java 7 - traditional try-catch
        String file6 = "java6file.out";
        try {
            OutputStream outputStream = new FileOutputStream(file6);
            outputStream.write("This is sample".getBytes());
        } catch (StringIndexOutOfBoundsException | IOException e) {
            System.out.println("error on file6 " + e.getMessage());
        }
        // =========== with java 7 ==============================

        String file7 = "java7file.out";
        try (OutputStream outputStream = new FileOutputStream(file7)) {
            outputStream.write("This is sample".getBytes());
        } catch (StringIndexOutOfBoundsException | IOException e) {
            System.out.println("error on file7 " + e.getMessage());
        }
        // =========== results ==============================
        File file = new File(file6);
        if (file.delete()) {
            System.out.println("File Deleted - " + file6);
        } else {
            System.out.println("File Deletion Error - " + file6);
        }
        File file2 = new File(file7);
        if (file2.delete()) {
            System.out.println("File Deleted - " + file7);
        } else {
            System.out.println("File Deletion Error - " + file7);
        }
    }
}
```

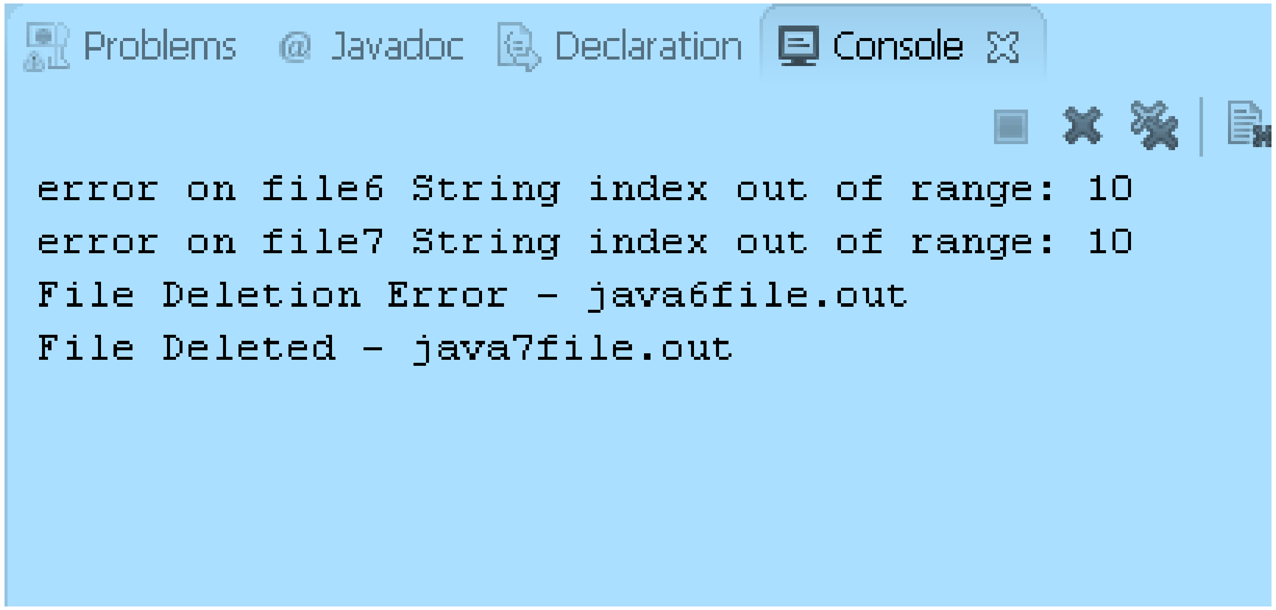


```
File Deletion Error - java6file.out
File Deleted - java7file.out
```

Try with Resources on Exception Example



```
public class TryWithResourceOnException {
    public static void main(String[] args) {
        // Before java 7 - traditional try-catch
        String file6 = "java6file.out";
        try {
            OutputStream outputStream = new FileOutputStream(file6);
            outputStream.write("This is sample".getBytes());
            "virtusa".charAt(10);
        } catch (StringIndexOutOfBoundsException | IOException e) {
            System.out.println("error on file6 " + e.getMessage());
        }
        // =========== with java 7 ===============================
        String file7 = "java7file.out";
        try (OutputStream outputStream = new FileOutputStream(file7)) {
            outputStream.write("This is sample".getBytes());
            "virtusa".charAt(10);
        } catch (StringIndexOutOfBoundsException | IOException e) {
            System.out.println("error on file7 " + e.getMessage());
        }
        // =========== results ==============================
        File file = new File(file6);
        if (file.delete()) {
            System.out.println("File Deleted - " + file6);
        } else {
            System.out.println("File Deletion Error - " + file6);
        }
        File file2 = new File(file7);
        if (file2.delete()) {
            System.out.println("File Deleted - " + file7);
        } else {
            System.out.println("File Deletion Error - " + file7);
        }
    }
}
```



```
error on file6 String index out of range: 10
error on file7 String index out of range: 10
File Deletion Error - java6file.out
File Deleted - java7file.out
```

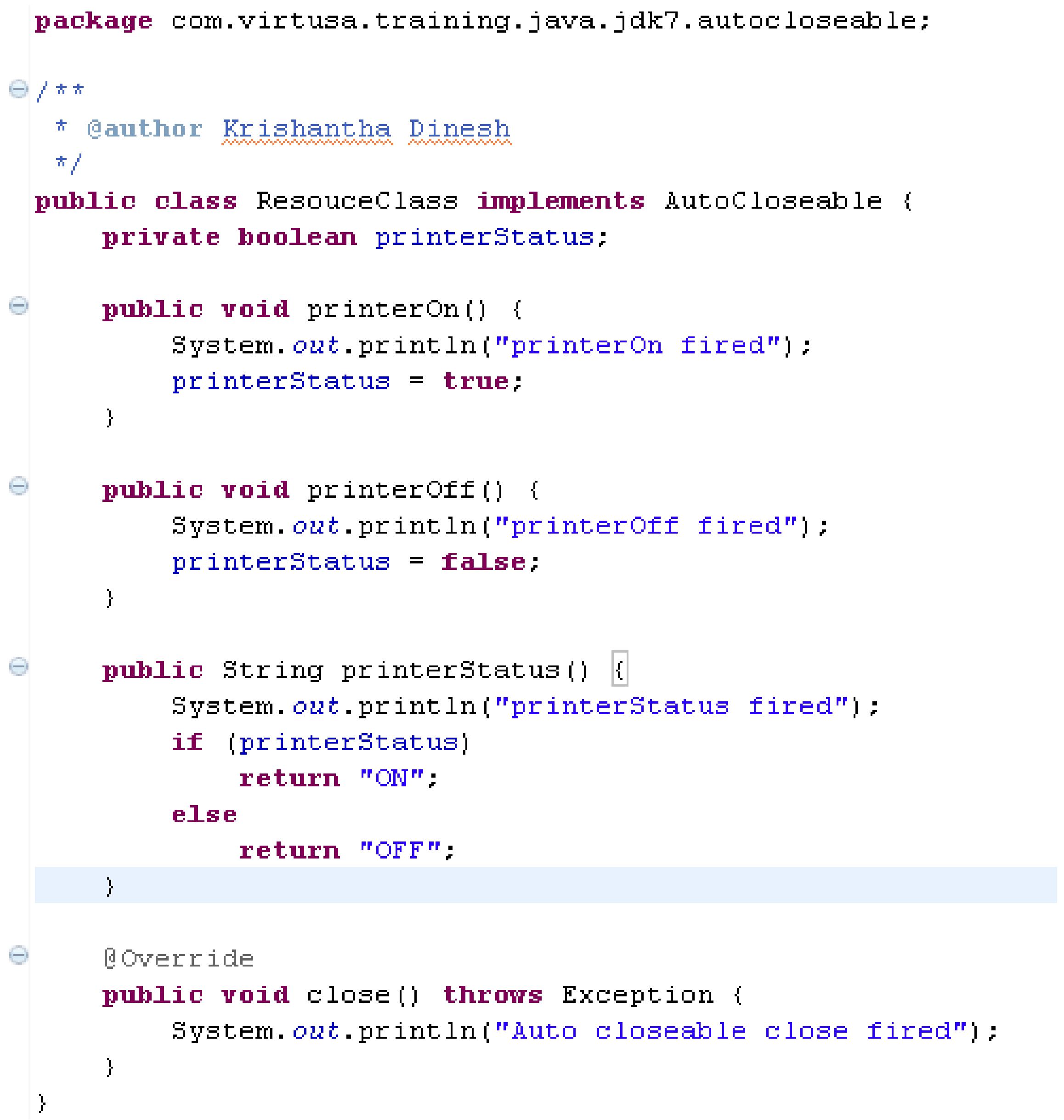
Autocloseable

- A new interface AutoCloseable is introduced
- Existing Closeable interface is changed to extend AutoCloseable interface
- A new method addSuppressed(Exception) is added to Throwable
- Exceptions throwen from close method of AutoCloseable are suppressed in favor of exceptions throwed from try-catch block
- It is simple interface as

```
public interface AutoClosable {
public void close() throws Exception;
}
```

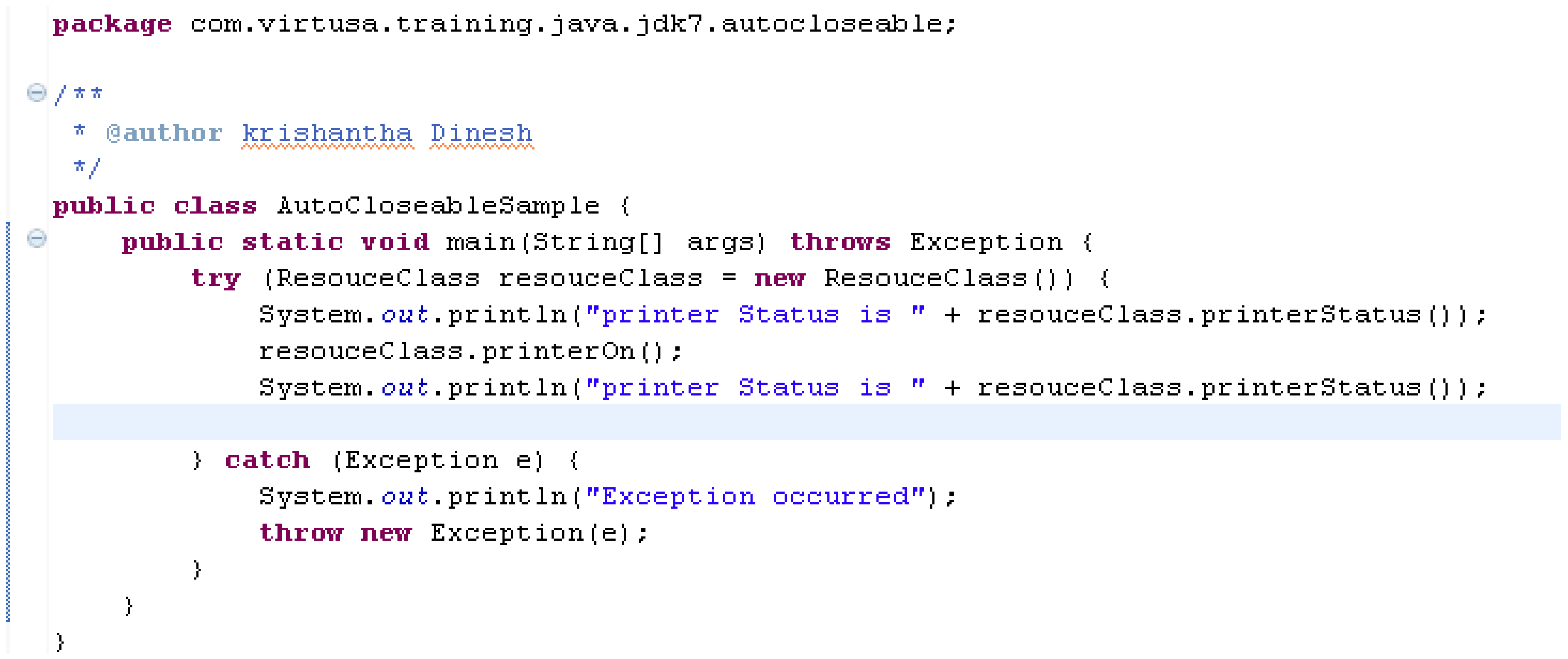
- Read JavaDoc of Autocloseable for more detail ☺

Autoclosable Example [resource]

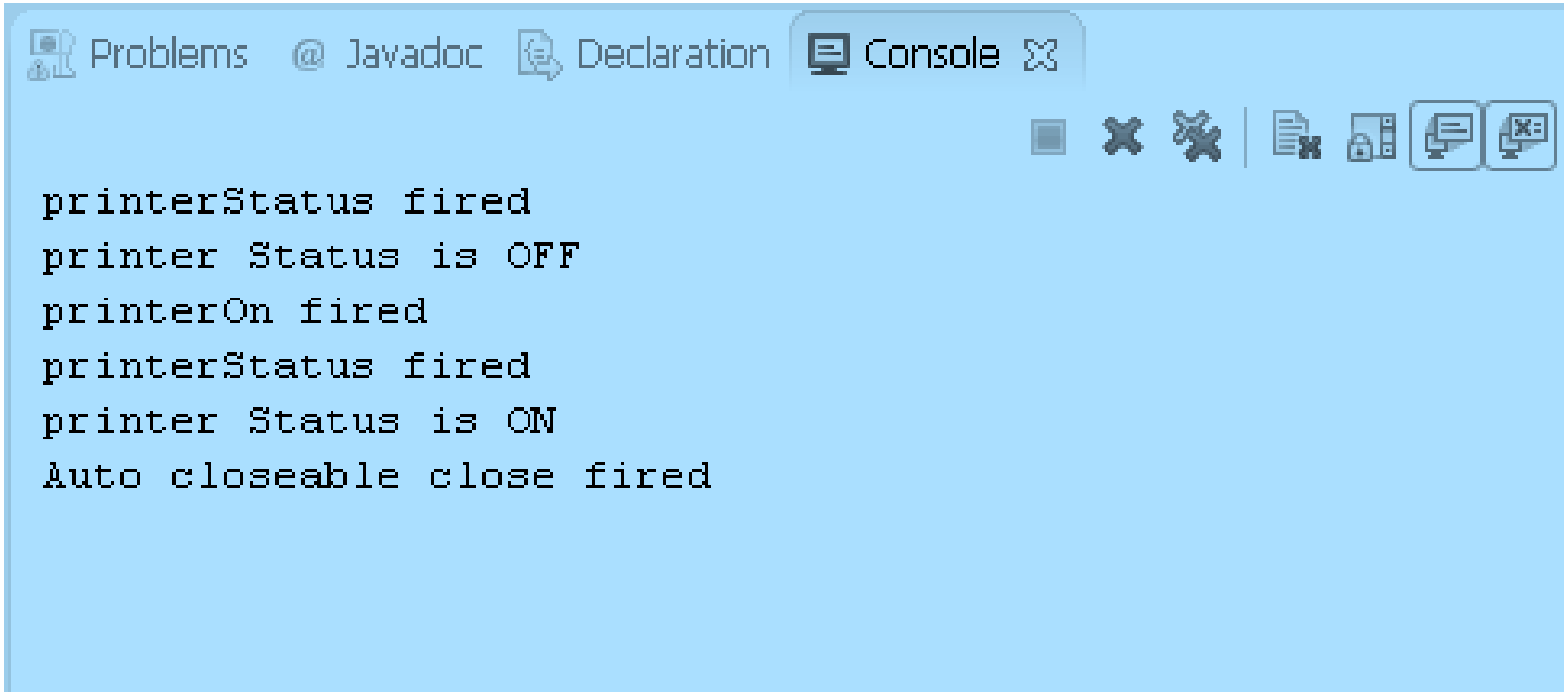


```java
package com.virtusa.training.java.jdk7.autocloseable;

/**
 * @author Krishantha Dinesh
 */
public class ResouceClass implements AutoCloseable {
    private boolean printerStatus;

    public void printerOn() {
        System.out.println("printerOn fired");
        printerStatus = true;
    }

    public void printerOff() {
        System.out.println("printerOff fired");
        printerStatus = false;
    }

    public String printerStatus() {
        System.out.println("printerStatus fired");
        if (printerStatus)
            return "ON";
        else
            return "OFF";
    }

    @Override
    public void close() throws Exception {
        System.out.println("Auto closeable close fired");
    }
}
```

Autoclosable Example [implementation]



```
package com.virtusa.training.java.jdk7.autocloseable;

/**
 * @author krishantha Dinesh
 */
public class AutoCloseableSample {
    public static void main(String[] args) throws Exception {
        try (ResouceClass resouceClass = new ResouceClass()) {
            System.out.println("printer Status is " + resouceClass.printerStatus());
            resouceClass.printerOn();
            System.out.println("printer Status is " + resouceClass.printerStatus());

        } catch (Exception e) {
            System.out.println("Exception occurred");
            throw new Exception(e);
        }
    }
}
```

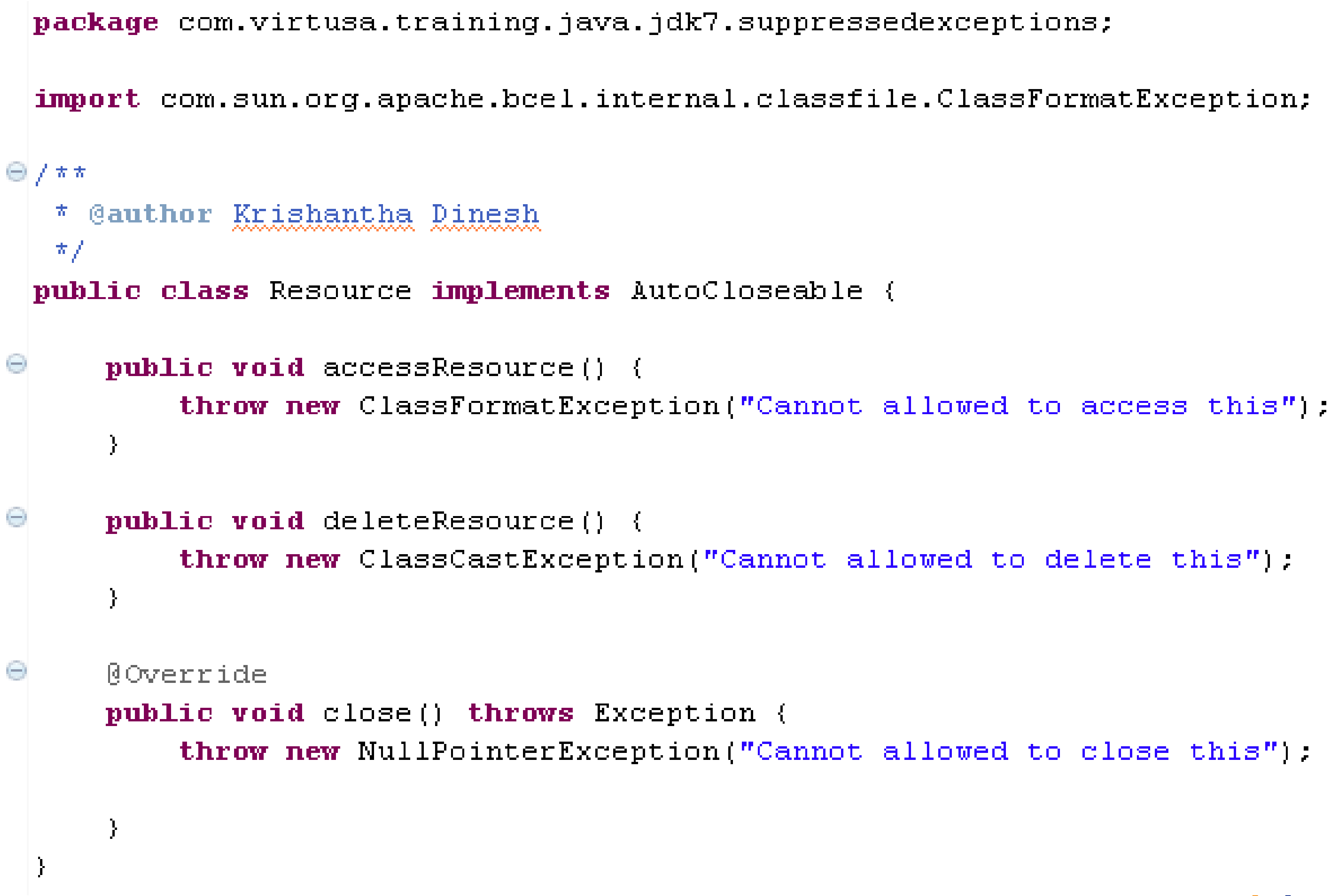


```
printerStatus fired
printer Status is OFF
printerOn fired
printerStatus fired
printer Status is ON
Auto closeable close fired
```

Suppressed exception

- It means exceptions thrown in the code but were ignored somehow
- encountering suppressed exceptions is when a try-with-resources statement encounters an exception within the try block and then encounters another exception in implicitly trying to close the related resource
- Because multiple exceptions may occur while closing AutoCloseable resources, additional exceptions are attached to a primary exception as suppressed exceptions.
- A new constructor and two new methods were added to the Throwable class (parent of Exception and Error classes) in JDK 7. These are as below
  - Throwable.getSupressed(); // Returns Throwable[]
  - Throwable.addSupressed(aThrowable);

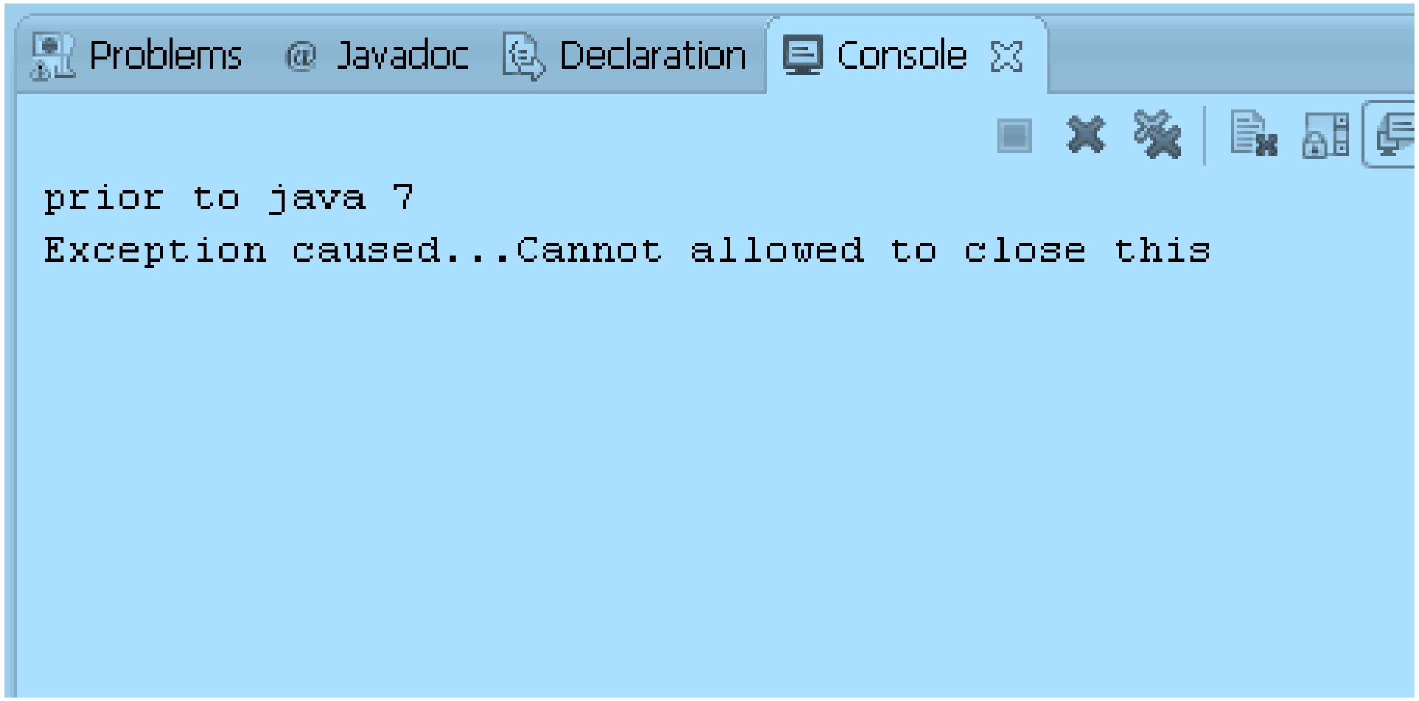
Suppressed exception - example



```
package com.virtusa.training.java.jdk7.suppressedexceptions;

import com.sun.org.apache.bcel.internal.classfile.ClassFormatException;

/**
 * @author Krishantha Dinesh
 */
public class Resource implements AutoCloseable {

    public void accessResource() {
        throw new ClassFormatException("Cannot allowed to access this");
    }

    public void deleteResource() {
        throw new ClassCastException("Cannot allowed to delete this");
    }

    @Override
    public void close() throws Exception {
        throw new NullPointerException("Cannot allowed to close this");

    }
}
```

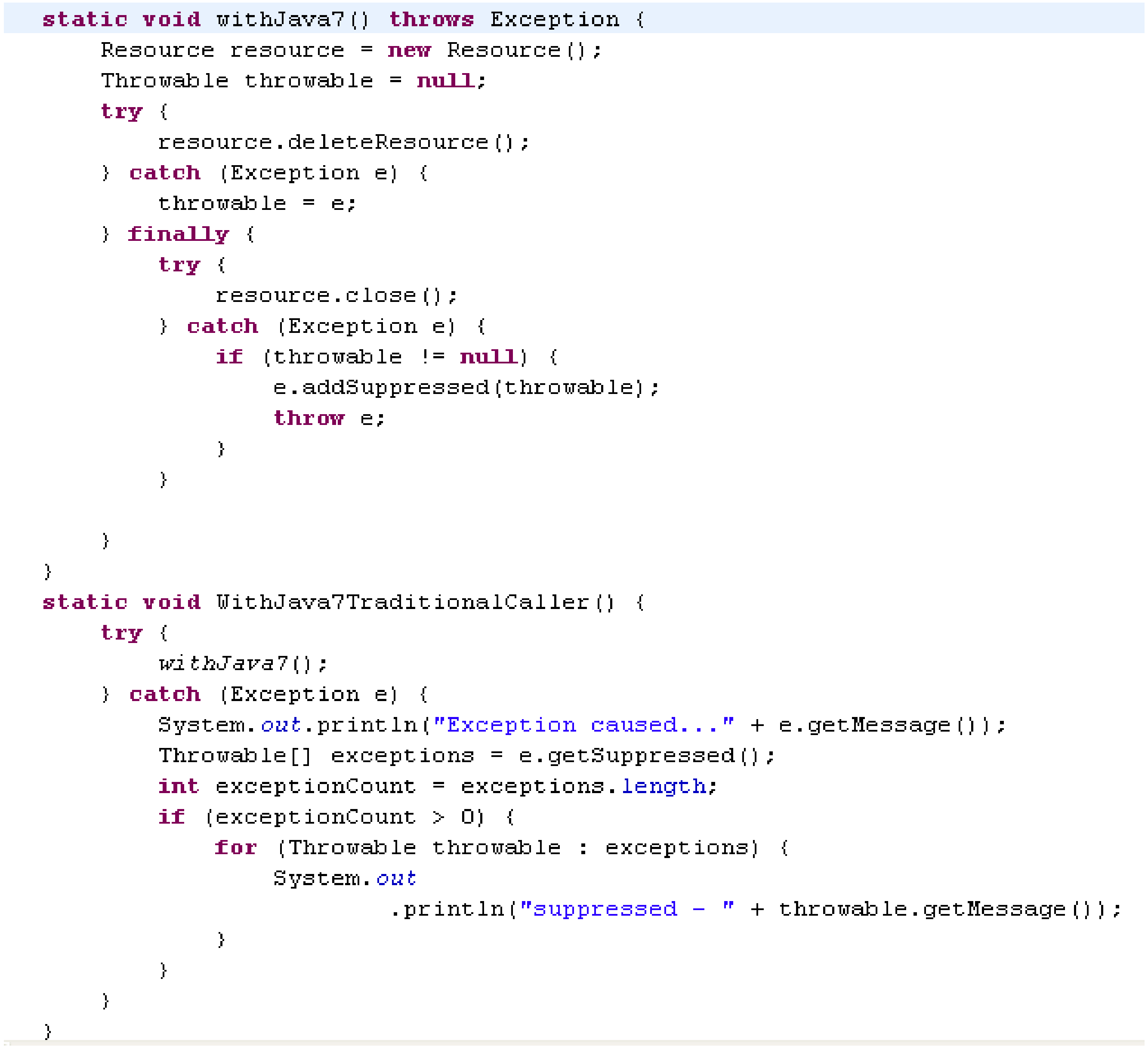
Suppressed exception – example Before java 7

```
static void priorToJava7() throws Exception {
    Resource resource = new Resource();
    try {
        resource.deleteResource();
    } finally {
        resource.close();
    }
}

static void priorToJava7Caller() {
    try {
        priorToJava7();
    } catch (Exception e) {
        System.out.println("Exception caused..." + e.getMessage());
        Throwable[] exceptions = e.getSuppressed();
        int exceptionCount = exceptions.length;
        if (exceptionCount > 0) {
            for (Throwable throwable : exceptions) {
                System.out
                        .println("suppressed - " + throwable.getMessage());
            }
        }
    }
}
```



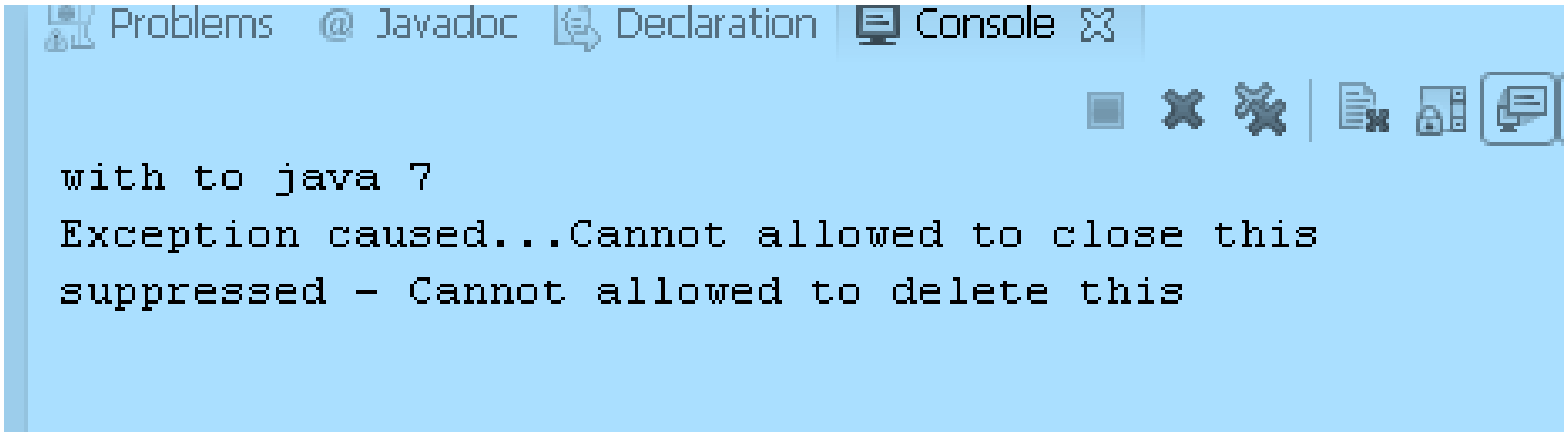
```
prior to java 7
Exception caused...Cannot allowed to close this
```

Suppressed exception – example with java 7



```java
static void withJava7() throws Exception {
    Resource resource = new Resource();
    Throwable throwable = null;
    try {
        resource.deleteResource();
    } catch (Exception e) {
        throwable = e;
    } finally {
        try {
            resource.close();
        } catch (Exception e) {
            if (throwable != null) {
                e.addSuppressed(throwable);
                throw e;
            }
        }

    }
}
static void WithJava7TraditionalCaller() {
    try {
        withJava7();
    } catch (Exception e) {
        System.out.println("Exception caused..." + e.getMessage());
        Throwable[] exceptions = e.getSuppressed();
        int exceptionCount = exceptions.length;
        if (exceptionCount > 0) {
            for (Throwable throwable : exceptions) {
                System.out
                        .println("suppressed - " + throwable.getMessage());
            }
        }
    }
}
```

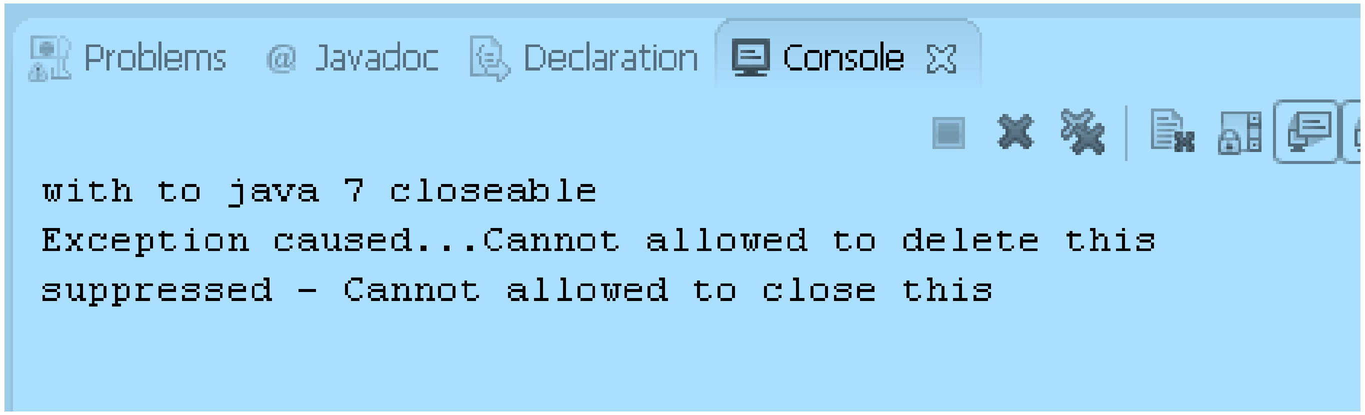
virtusa®



```
with to java 7
Exception caused...Cannot allowed to close this
suppressed - Cannot allowed to delete this
```

Suppressed exception – example with try with resource

```java
static void withJava7AutoCloseable() throws Exception {
    try (Resource resource = new Resource()) {
        resource.deleteResource();
    }
}

static void withJava7Caller() {

    try {
        withJava7AutoCloseable();
    } catch (Exception e) {
        System.out.println("Exception caused..." + e.getMessage());
        Throwable[] exceptions = e.getSuppressed();
        int exceptionCount = exceptions.length;
        if (exceptionCount > 0) {
            for (Throwable throwable : exceptions) {
                System.out
                        .println("suppressed - " + throwable.getMessage());
            }
        }
    }

}
```



```
with to java 7 closeable
Exception caused...Cannot allowed to delete this
suppressed - Cannot allowed to close this
```

NIO and NIO.2

- Java NIO -- the New Input/Output API package-- was introduced with J2SE 1.4 in 2002
- Java NIO's purpose was to improve the programming of I/O-intensive chores on the Java platform
- NIO boosts Java application performance by getting "closer to the metal“ [NIO and NIO.2 APIs expose lower-level-system operating-system (OS) entry points]
- The tradeoff of NIO is that it simultaneously gives us greater control over I/O and demands that we exercise more care than we would with basic I/O programming
- Another aspect of NIO is its attention to application expressivity

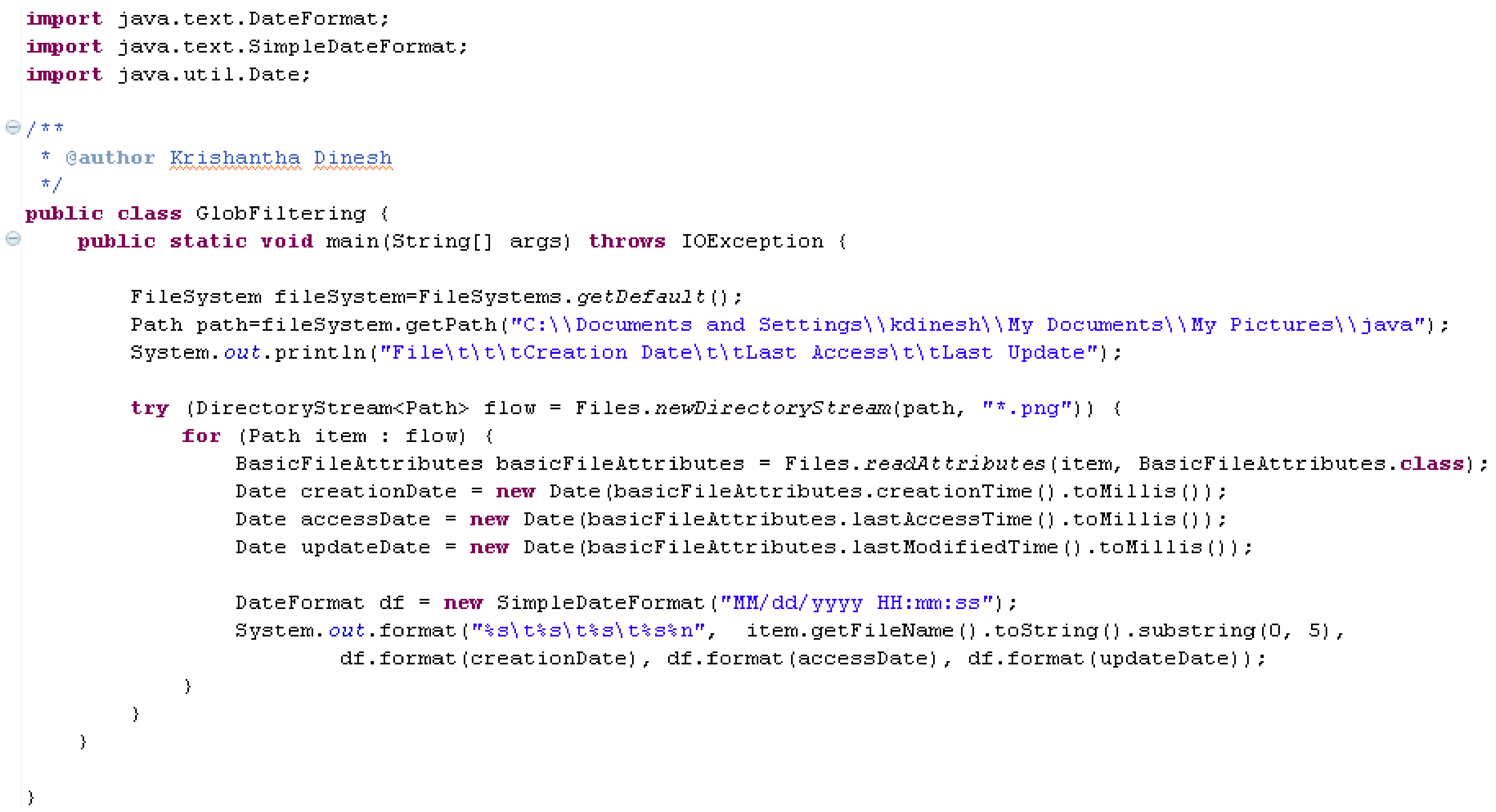
NIO and nio.2

- To address limitations, JSR 203 – More New I/O APIs for the Java Platform ("NIO.2") was proposed.
- The NIO.2 API adds three sub packages to java.nio:
  - java.nio.file: main package for NIO.2, defines interfaces and classes to access files and file systems using the new API;
  - java.nio.file.attribute: contains classes and interfaces related to reading and changing file attributes;
  - java.nio.file.spi: contains service provider interfaces for NIO.2, to be used by developers who wish the implement support to a new type of file system.

Java.nio.file

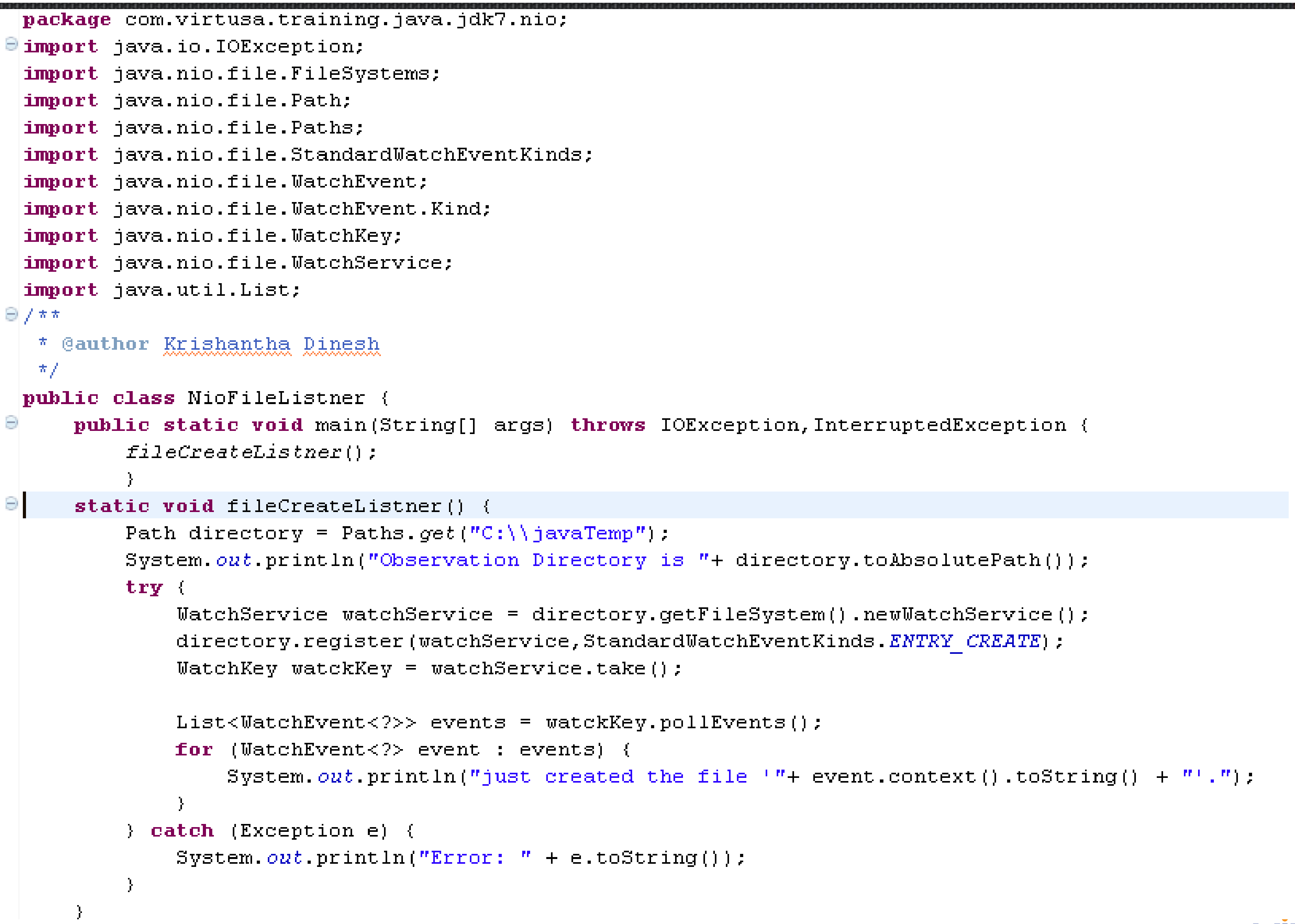
- The following classes (from package java.nio.file) represent the main concepts of the new API:
  - Path: immutable class that represents the path to any file
  - Files: class that contains several static methods for the execution of operations in files given their paths
  - FileSystem: class that represents the file system as a whole, used to obtain paths to files
  - FileSystems: class that contains several static methods for obtaining a file system. The method FileSystems.getDefault() obtains an object that allows us to access all files to which the JVM has access
  - FileStore: class that represents the file storing mechanism that is manipulated by the different methods of the API (a partition of a hard drive, an external device plugged in via USB, etc.).

NIO.2 – Glob filtering [what use in shell and cmd]



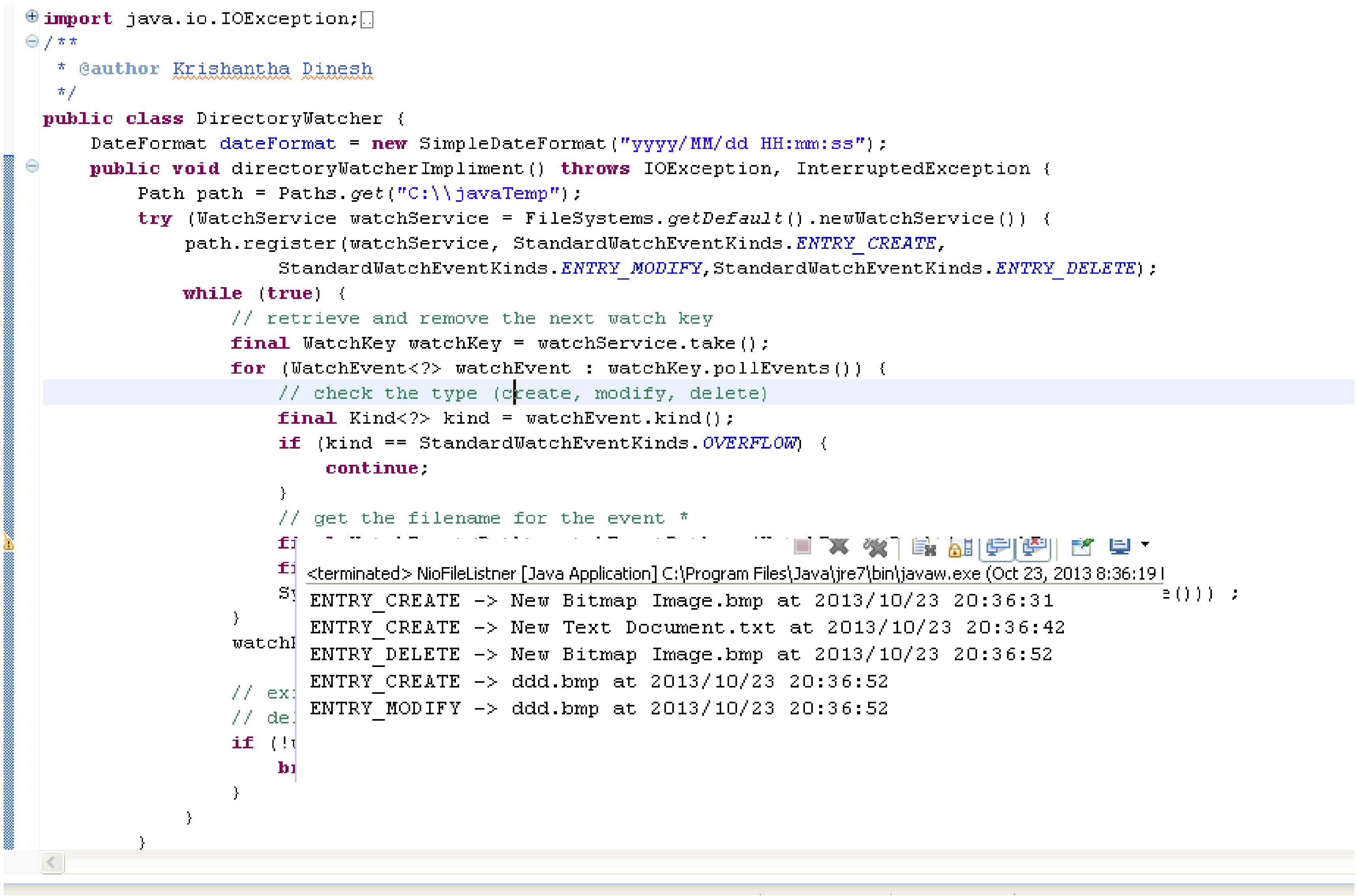
```java
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * @author Krishantha Dinesh
 */
public class GlobFiltering {
    public static void main(String[] args) throws IOException {

        FileSystem fileSystem=FileSystems.getDefault();
        Path path=fileSystem.getPath("C:\\Documents and Settings\\kdinesh\\My Documents\\My Pictures\\java");
        System.out.println("File\t\t\tCreation Date\t\tLast Access\t\tLast Update");

        try (DirectoryStream<Path> flow = Files.newDirectoryStream(path, "*.png")) {
            for (Path item : flow) {
                BasicFileAttributes basicFileAttributes = Files.readAttributes(item, BasicFileAttributes.class);
                Date creationDate = new Date(basicFileAttributes.creationTime().toMillis());
                Date accessDate = new Date(basicFileAttributes.lastAccessTime().toMillis());
                Date updateDate = new Date(basicFileAttributes.lastModifiedTime().toMillis());

                DateFormat df = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");
                System.out.format("%s\t%s\t%s\t%s%n",  item.getFileName().toString().substring(0, 5),
                        df.format(creationDate), df.format(accessDate), df.format(updateDate));
            }
        }
    }

}
```

NIO.2 example [watch for new file creation]



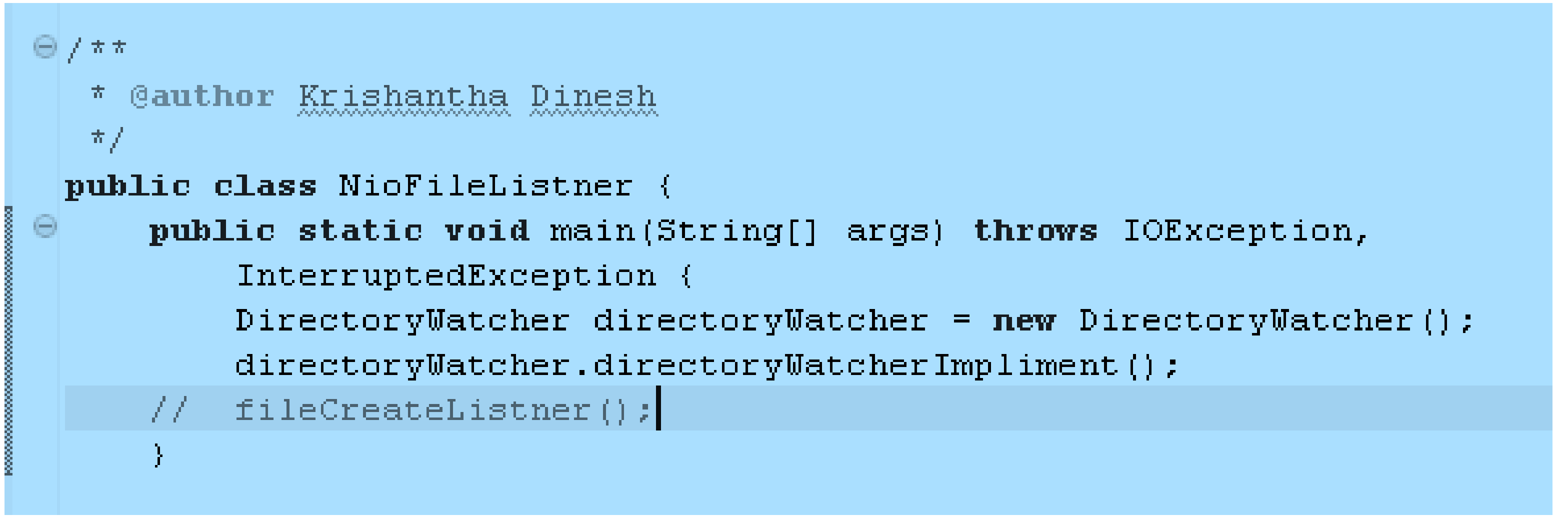
```java
package com.virtusa.training.java.jdk7.nio;
import java.io.IOException;
import java.nio.file.FileSystems;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardWatchEventKinds;
import java.nio.file.WatchEvent;
import java.nio.file.WatchEvent.Kind;
import java.nio.file.WatchKey;
import java.nio.file.WatchService;
import java.util.List;
/**
 * @author Krishantha Dinesh
 */
public class NioFileListner {
    public static void main(String[] args) throws IOException,InterruptedException {
        fileCreateListner();
        }
    static void fileCreateListner() {
        Path directory = Paths.get("C:\\javaTemp");
        System.out.println("Observation Directory is "+ directory.toAbsolutePath());
        try {
            WatchService watchService = directory.getFileSystem().newWatchService();
            directory.register(watchService,StandardWatchEventKinds.ENTRY_CREATE);
            WatchKey watckKey = watchService.take();

            List<WatchEvent<?>> events = watckKey.pollEvents();
            for (WatchEvent<?> event : events) {
                System.out.println("just created the file '"+ event.context().toString() + "'.");
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.toString());
        }
    }
```

Directory Watcher - Advance



```
import java.io.IOException;
/**
 * @author Krishantha Dinesh
 */
public class DirectoryWatcher {
    DateFormat dateformat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
    public void directoryWatcherImpliment() throws IOException, InterruptedException {
        Path path = Paths.get("C:\\javaTemp");
        try (WatchService watchService = FileSystems.getDefault().newWatchService()) {
            path.register(watchService, StandardWatchEventKinds.ENTRY_CREATE,
                    StandardWatchEventKinds.ENTRY_MODIFY,StandardWatchEventKinds.ENTRY_DELETE);
            while (true) {
                // retrieve and remove the next watch key
                final WatchKey watchKey = watchService.take();
                for (WatchEvent<?> watchEvent : watchKey.pollEvents()) {
                    // check the type (create, modify, delete)
                    final Kind<?> kind = watchEvent.kind();
                    if (kind == StandardWatchEventKinds.OVERFLOW) {
                        continue;
                    }
                    // get the filename for the event *
                    fi
                    fi
                    Sy                                                                        e())) ;
                }
                watchK

                // ex
                // de
                if (!w
                    br
                }
            }
        }
```

```
<terminated> NioFileListner [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Oct 23, 2013 8:36:19 
ENTRY_CREATE -> New Bitmap Image.bmp at 2013/10/23 20:36:31
ENTRY_CREATE -> New Text Document.txt at 2013/10/23 20:36:42
ENTRY_DELETE -> New Bitmap Image.bmp at 2013/10/23 20:36:52
ENTRY_CREATE -> ddd.bmp at 2013/10/23 20:36:52
ENTRY_MODIFY -> ddd.bmp at 2013/10/23 20:36:52
```

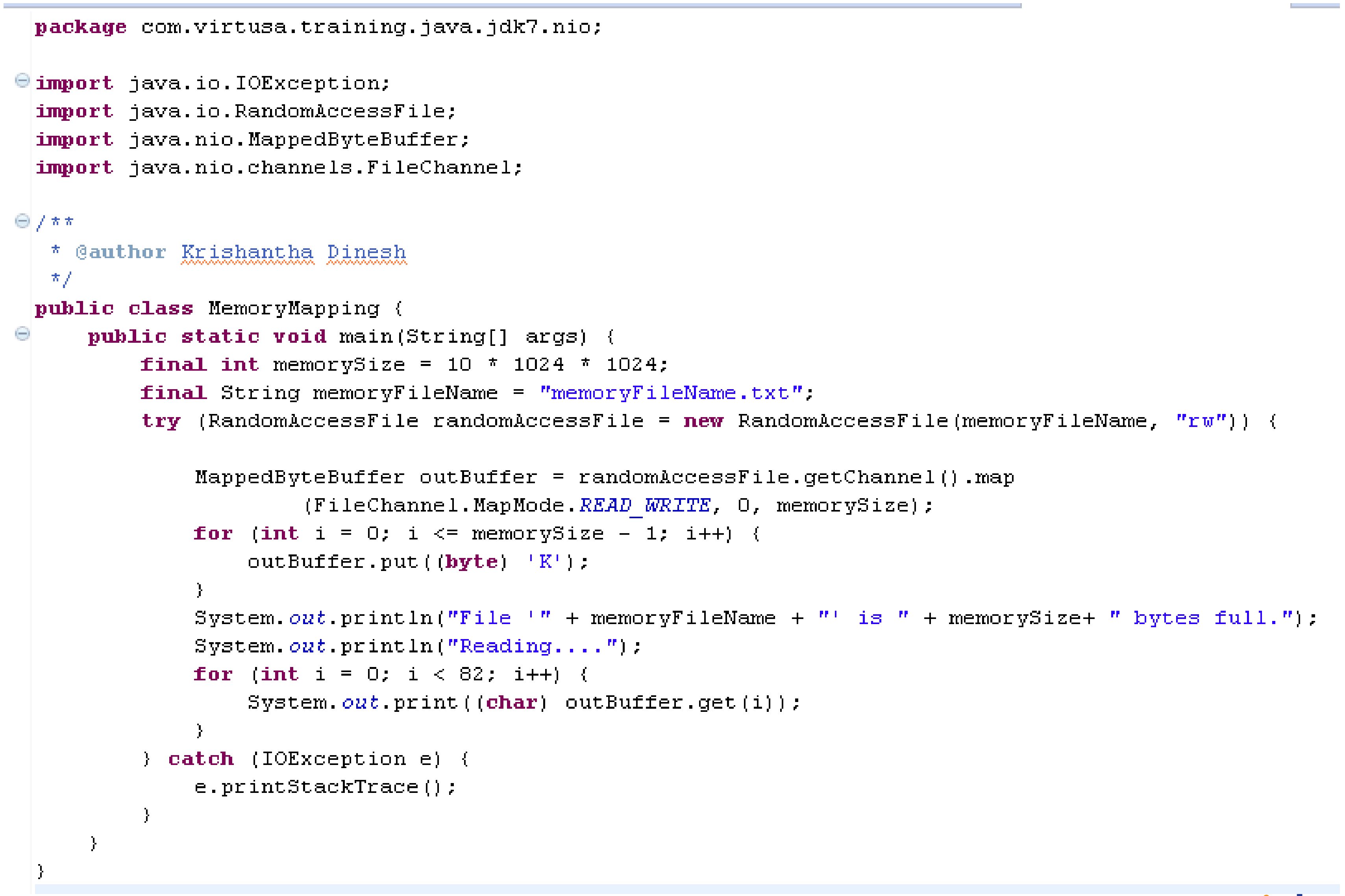


```
/**
 * @author Krishantha Dinesh
 */
public class NioFileListner {
    public static void main(String[] args) throws IOException,
        InterruptedException {
        DirectoryWatcher directoryWatcher = new DirectoryWatcher();
        directoryWatcher.directoryWatcherImpliment();
    //  fileCreateListner();
    }
```

Memory Mapping

- The most consistently dramatic performance improvement in the use of NIO involves memory mapping
- Memory mapping has a number of consequences and implications
- memory mapping is interesting not only for the raw speed of I/O, but also because several different readers and writers can attach simultaneously to the same file image
- This technique is powerful enough to be dangerous

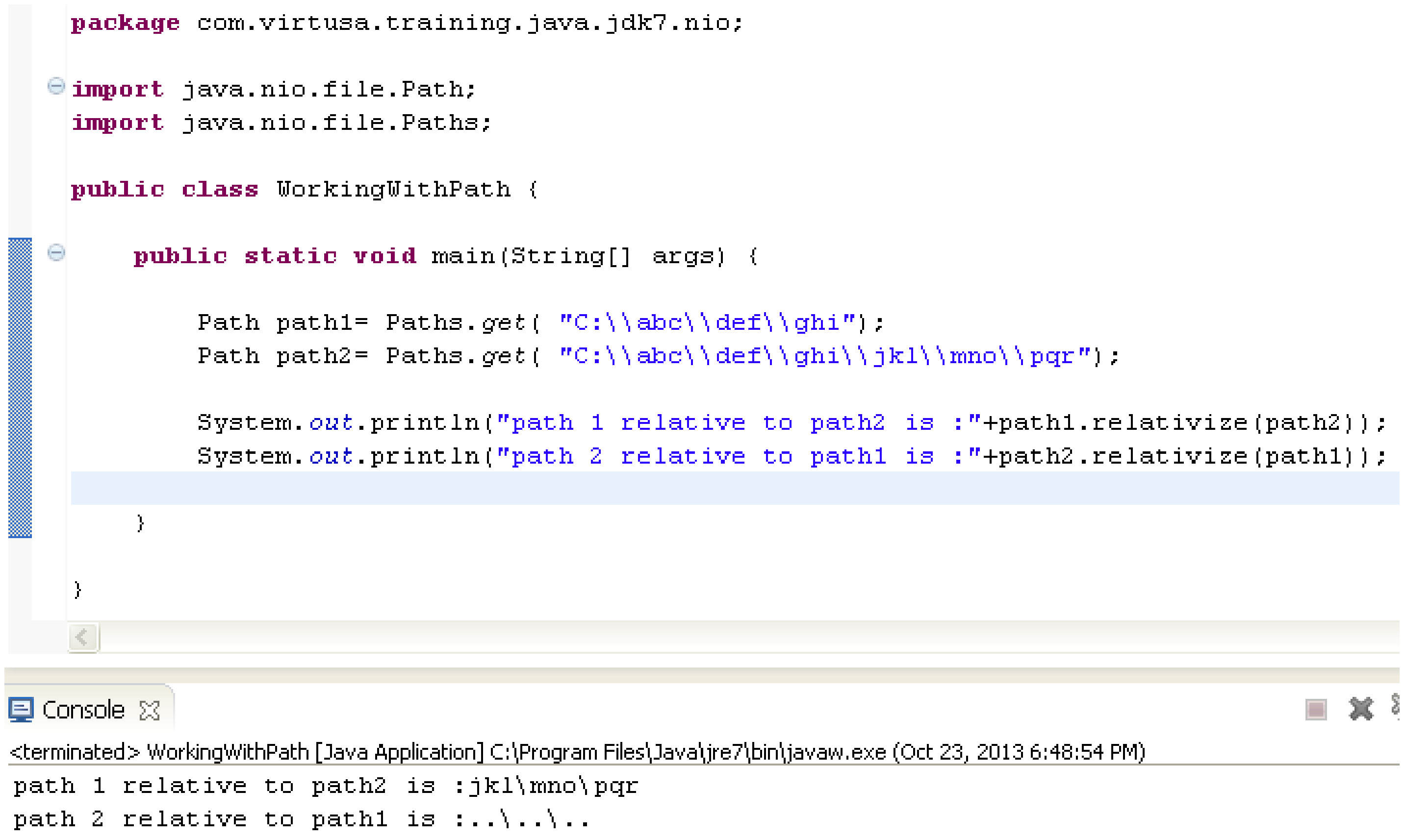
NIO - channel



```java
package com.virtusa.training.java.jdk7.nio;

import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

/**
 * @author Krishantha Dinesh
 */
public class MemoryMapping {
    public static void main(String[] args) {
        final int memorySize = 10 * 1024 * 1024;
        final String memoryFileName = "memoryFileName.txt";
        try (RandomAccessFile randomAccessFile = new RandomAccessFile(memoryFileName, "rw")) {

            MappedByteBuffer outBuffer = randomAccessFile.getChannel().map
                    (FileChannel.MapMode.READ_WRITE, 0, memorySize);
            for (int i = 0; i <= memorySize - 1; i++) {
                outBuffer.put((byte) 'K');
            }
            System.out.println("File '" + memoryFileName + "' is " + memorySize+ " bytes full.");
            System.out.println("Reading....");
            for (int i = 0; i < 82; i++) {
                System.out.print((char) outBuffer.get(i));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Working with Path

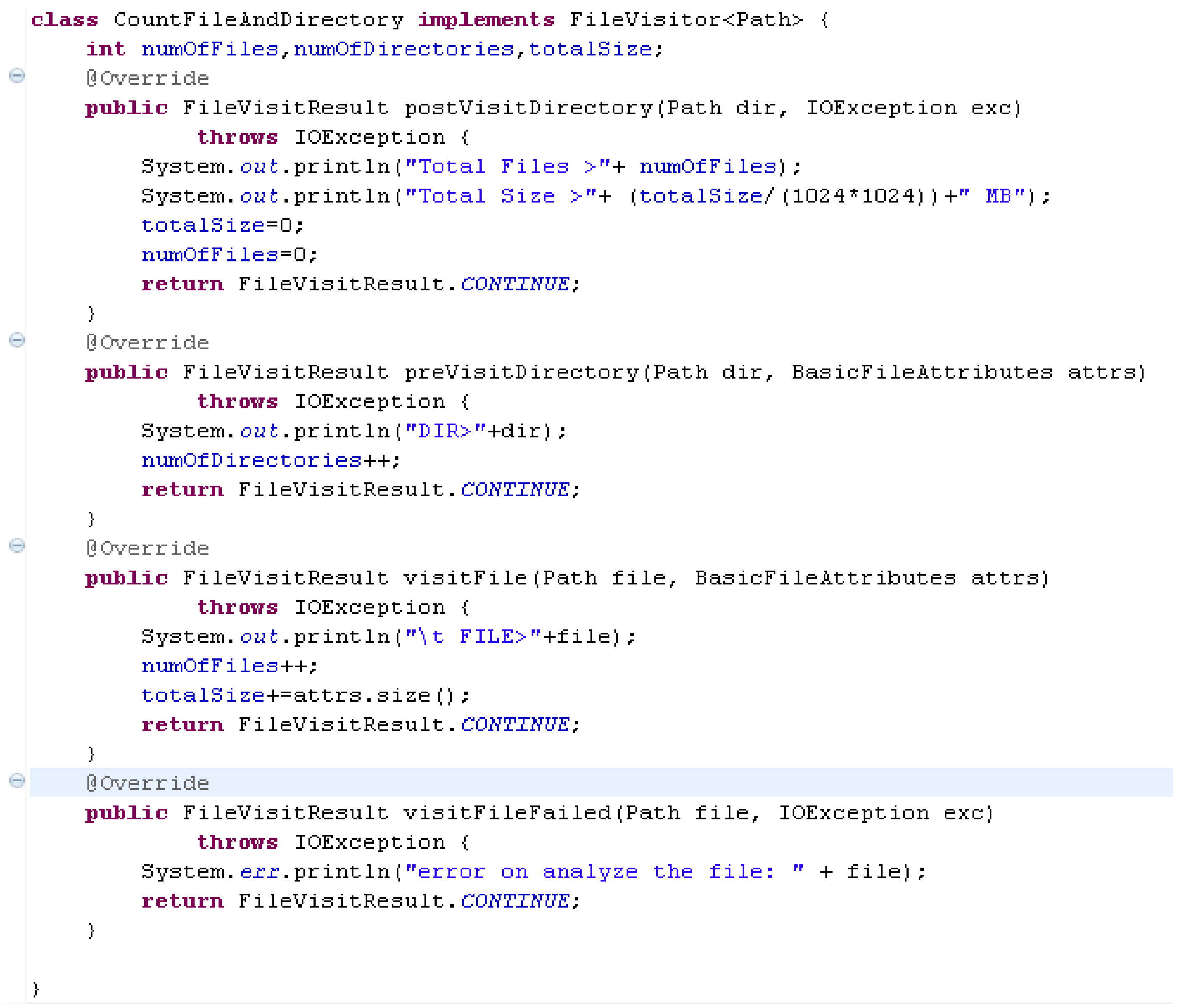
- Resolve path relative to other path



```
package com.virtusa.training.java.jdk7.nio;

import java.nio.file.Path;
import java.nio.file.Paths;

public class WorkingWithPath {

    public static void main(String[] args) {

        Path path1= Paths.get( "C:\\abc\\def\\ghi");
        Path path2= Paths.get( "C:\\abc\\def\\ghi\\jkl\\mno\\pqr");

        System.out.println("path 1 relative to path2 is :"+path1.relativize(path2));
        System.out.println("path 2 relative to path1 is :"+path2.relativize(path1));

    }

}
```

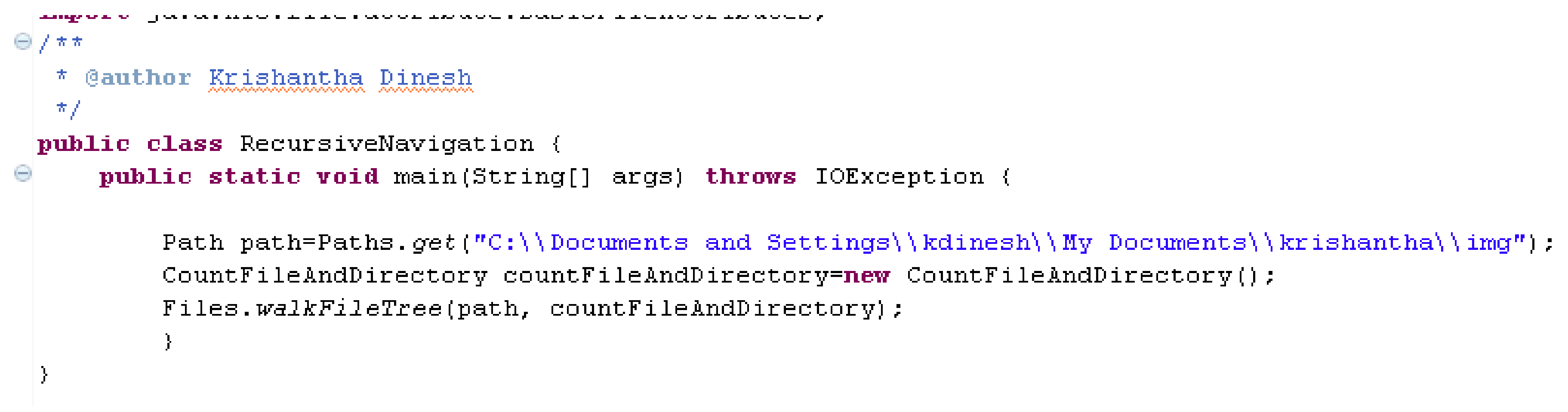
```
<terminated> WorkingWithPath [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Oct 23, 2013 6:48:54 PM)
path 1 relative to path2 is :jkl\mno\pqr
path 2 relative to path1 is :..\..\..
```

NIO.2 - recursive navigation

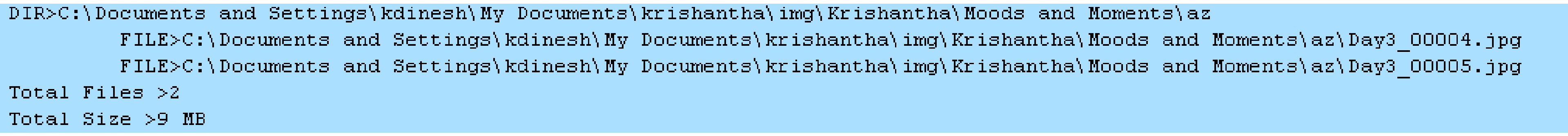


```java
class CountFileAndDirectory implements FileVisitor<Path> {
    int numOfFiles,numOfDirectories,totalSize;
    @Override
    public FileVisitResult postVisitDirectory(Path dir, IOException exc)
            throws IOException {
        System.out.println("Total Files >"+ numOfFiles);
        System.out.println("Total Size >"+ (totalSize/(1024*1024))+" MB");
        totalSize=0;
        numOfFiles=0;
        return FileVisitResult.CONTINUE;
    }
    @Override
    public FileVisitResult preVisitDirectory(Path dir, BasicFileAttributes attrs)
            throws IOException {
        System.out.println("DIR>"+dir);
        numOfDirectories++;
        return FileVisitResult.CONTINUE;
    }
    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attrs)
            throws IOException {
        System.out.println("\t FILE>"+file);
        numOfFiles++;
        totalSize+=attrs.size();
        return FileVisitResult.CONTINUE;
    }
    @Override
    public FileVisitResult visitFileFailed(Path file, IOException exc)
            throws IOException {
        System.err.println("error on analyze the file: " + file);
        return FileVisitResult.CONTINUE;
    }

}
```

NIO.2 - recursive navigation - Caller



```
import java.nio.file.attribute.BasicFileAttributes;
/**
 * @author Krishantha Dinesh
 */
public class RecursiveNavigation {
    public static void main(String[] args) throws IOException {

        Path path=Paths.get("C:\\Documents and Settings\\kdinesh\\My Documents\\krishantha\\img");
        CountFileAndDirectory countFileAndDirectory=new CountFileAndDirectory();
        Files.walkFileTree(path, countFileAndDirectory);
        }
}
```



```
DIR>C:\Documents and Settings\kdinesh\My Documents\krishantha\img\Krishantha\Moods and Moments\az
         FILE>C:\Documents and Settings\kdinesh\My Documents\krishantha\img\Krishantha\Moods and Moments\az\Day3_00004.jpg
         FILE>C:\Documents and Settings\kdinesh\My Documents\krishantha\img\Krishantha\Moods and Moments\az\Day3_00005.jpg
Total Files >2
Total Size >9 MB
```

Happy Coding ☺ with