

JAVA2EE TUTORIAL

JAVA2 Tutorial Requirements

- This tutorial assumes that you have taken computer science classes through CS 2308 or have equivalent experience. If you do not have either of these, this tutorial may not be well suited for you.
- Knowledge of C++ and object oriented design will also be extremely helpful.

TABLE OF CONTENTS

1. Why Use Java?.....	4
2. Basic Java Variables Types and Operators.....	5
3. Java Syntax Primer	8
4. Java Program Flow Control:	10
5. Key Differences between C++ and Java:.....	16
6. Java Compilation Basics	17
7. Advanced Java Topics	20
8. Useful Links For Java:	25
9. Code Appendix	26

1. Why Use Java?

There are three main reasons for using the JAVA programming language over other choices: portability, simplicity of coding, and widespread use; especially in web development.

Java is completely portable between different operating systems (Mac, Linux, and Windows). This is because the code you write is only partially finished after it is compiled by the programmer. The rest of the work is performed on the user's computer using the Java Virtual Machine (JVM).

There is a JVM for each major operating system, and after being downloaded and installed by the user, it completes the Java code that the programmer wrote. The JVM adds in commands specific to the operating system it is installed on to make the program run correctly for that operating system. So the JVM for a Macintosh will add in Macintosh specific code and the JVM for Windows will add in Windows specific code.

This portability has led to Java being used widely, especially for the web development industry. Since web based applications run on multiple operating systems it was essential to have a programming language, such as Java, that could run on any operating system. Today Java is everywhere, and is used for most major web applications.

Another factor in Java's increased use is its simplicity of programming in comparison to other languages, especially C++. Java enforces error checking, eliminates pointers, cannot be written without using object oriented design, and prevents bad programming practices by building these functions directly into Java's grammar. These features make Java a much simpler (and much less error prone) than many other languages that do not have these restrictions.

2. Basic Java Variables Types and Operators

Basic Variable Types:

Variable Type	Declaration Example	Memory Range / Notes
Byte	byte register = 64;	(8 bit = -2^8 to $2^8 - 1$)
Short Integer	short shortInt = 24859;	(16 bit = -2^{16} to $2^{16} - 1$)
Integer	int x = 20;	(32 bit = -2^{32} to $2^{32} - 1$)
Long Integer	longInt = 3;	(64 bit = -2^{64} to $2^{64} - 1$)
Boolean	boolean loopDone = false;	0 or 1
Float	float temperature = 34.6;	$1.4E-45$ to $3.4E+38$
Double	double speed = 5945.32;	$4.9E-324$ to $1.7E+308$
Final <variable type>	final float PI = 3.141592	Same as const in C++
String	String lastName = "Koh";	Actually an object.

Declaring Arrays

```
int[ ] myArray;  
int myArray[ ];  
myArray = new int[10];
```

```
int[ ] betterArray = new int[10];
```

```
int[ ] nextArray = { 1 , 32 , 4, 54 ,3 ,65 ,34 ,343 };
```

```
int[ ][ ] twiceTheArray;  
twiceTheArray = new int[10][10];
```

```
makeButtons( new String[] { "Back", "Next" } );
```

Casting can be done from one variable type to another variable type by placing the basic type in parentheses next to the variable:

```
int x = 65;  
char bigA = (char)x;  
System.out.println(bigA);
```

This produces the output 'A' because 65 is the ASCII code for capital a.

Casting can also be done from object to object, object to primitive, and primitive to object, but these types of casts are beyond the scope of this tutorial.

Java Operators and Order of Operations:

.	[]	()					
++	--	!	~	instanceof			
new							
*	/	%					
+	-						
<<	>>	>>>					
<	>	<=	>=				
==	!=						
&							
^							
&&							
?:	=	+=	--+	*=	/=	%=	^=
&=	=	<<=	>>=	>>>=			

Operator Notes:

New allocates memory to a variable. Examples of this were given in the arrays section above, and will be used in example code section.

instanceof performs a test to see if a variable or object is the same type as a class of built in variable type. The instanceof test returns true below because x is an integer.

```
int x;
if (x instanceof int)
    System.out.println("x is an integer!");
```

?: Is a shorthand version of if-then-else for **returning values**:
 <test> ? <trueResult> : <falseResult>

The two segments of code listed below are identical:

```
int x = 2;
```

```
if ( x == 3 )
  x =0;
else
  x=x+1;
```

```
int x = 2;
x = x==3 ? x = 0 : x+1;
```

Difference between |,& and ||,&&

The primary difference is in how much work Java does in evaluating an expression. With & and | both expressions on either side of the operator are evaluated before a result is returned.

With || if a true result is found in the left expression or with && if a false result is found in the left expression, the remaining expression is not evaluated.

Consider:

```
int x = 2;
```

```
(x == 1)    &&    (x == 2)    &&    (x==3)    &&    (x==4)
(x == 1)    &     (x == 2)    &     (x==3)    &     (x==4)
```

The first line will evaluate `x == 1` and return false.

The second line will evaluate all four expressions and then return false.

3. Java Syntax Primer

Comments

Comments are done like C++:

```
// I'm commented
/* I am also commented
   as well */
```

Import Command

The import command is used like the include command to add code libraries and modules to your program.

In C++ you would have:

```
#include <iostream.h> or using std namespace;
```

In Java:

```
import java.io.*;
```

This java command will import all of the libraries and sub-libraries of java.io.

Strings

In Java, strings can be used in a very flexible manner. Below are all examples of legal uses of strings in Java:

```
String stringA;
String stringB;
String output;
int three = 3;

stringA = "a";
stringB = stringA + "b" + "c";
three = 123;
System.out.println(three + " " + stringB);
```

This snippet of code will result in the output: 123 abc

Classes and Methods

Classes are the fundamental building block of Java. They typically work very similarly to C++ classes with a few exceptions. First, all methods (functions) must be contained within a class. Even the main method is contained within a class. The class containing the main method is the class where the program will originate. The template for declaring classes and methods listed below. Code examples will be given later.

```
class <class name>
{
    <method 1 name>
    {
        code
    }

    public static void main(String arguments[])
    {
        code
    }
}
```

4. Java Program Flow Control:

Program flow control data structures should be very familiar to you, but they are here for completeness and due to minor differences between Java and other programming languages.

If-else

If-else statements are identical in syntax to C++ code. A few examples:

```
int x = 0;

if ( x == 0)
    {x = 3;}

else x++;

if ( x == 0)
    if ( y == 0)
        x=y=1;

else y=x;
```

Note that the “else y=x” is linked to the “if (y == 0)” statement because like C++ else statements always associate with the nearest if statement.

The one difference is that C++ if-else tests can return a number value while Java tests must return boolean values only.

```
if ( 3 )
    cout << “ Always true” << endl;
```

The conditional of 3 is fine for C++, but will cause an error in Java:

```
test.java:7: incompatible types
found   : int
required: boolean
    if ( 3 )
        ^
1 error
```

This is true of most all the flow control statements in this section that require a boolean value.

Switch

A switch statement has a syntax that is identical to C++ syntax. The general format for a switch statement is:

```
switch (expression) {  
    case constant:  
        code;  
        break;  
    case constant-2:  
        code;  
        break;  
    default:  
        default statements;  
}
```

For example:

```
int x = 6;  
  
switch (x)  
{  
    case 5:  
        System.out.println("x is 5");  
        break;  
  
    case 6:  
        System.out.println("x is 6");  
        break;  
  
    case 7:  
  
    default:  
        System.out.println("x is not 5 or 6");  
}
```

This example compares the value of x to 5, 6, and 7. If it is five or six, it prints out a statement saying this, and then exits the case statement (using break). If x equals seven it just continues to the default since there is no break statement to exit the switch. The default tells us that the x is not equal to five or six.

While and Do

These statements create loops in your program until a certain boolean condition has been violated. The standard formats are:

```
while ( Boolean condition )
{
    code
}
```

also:

```
do
{
    code
}
while ( boolean condition);
```

Note the semicolon at the end of the while statement when using a do-while loop.

Example:

```
int num = 0;
Boolean loopDone = false;
while ( !loopDone)
{
    num++;

    if (num == 10 )
        loopDone = true;
}
```

This code adds one to num until it reaches ten. At this point, loopDone becomes true making !loopDone false. (! is the not operator from section 2)

For

For loops work identically to C++ loops. The basic format is:

```
for ( initialize variable; condition; increment or decrement )
{
    code
}
```

For example:

```

int lastTime = 0;
int timeToGo = 0;

for ( timeToGo = 1; timeToGo < 3600; timeToGo++ )
{
    lastTime = timeToGo;
    timeToGo = timeToGo * 60;
}

```

This loop will repeat until timeToGo is greater than 3600.

Catch, Try, and Finally

Java is very sensitive about handling exceptions (errors) generated in a program. Common exceptions are going beyond the bounds of an array, or attempting to use a file that has not been opened. Exceptions by default are passed to the JVM which then terminates your program. To prevent your program from being killed by the JVM you must catch the exception yourself.

Also, these exceptions are divided into two types: mandatory and non-mandatory. Non-mandatory exceptions do not have to be explicitly handled by your code. Mandatory exceptions, however, must be handled by your code. You will get compile errors if you have not handled mandatory exceptions that occur in your code at run time.

For mandatory exceptions (and for good programming practice for non-mandatory exceptions) you have to place the code in a try block.

Examples:

```

int i = 0;

try {
    for( i = 0; i < 9; i++ )
        myArray[i] = i;
}

catch ( ArrayIndexOutOfBoundsException e )
{
    System.out.println("i went out of bounds for array myArray.");
}

catch ( NullPointerException e )
{
    System.out.print("Programming error! myArray doesn't exist: " + );
}

```

```
        System.out.println( e.getMessage() );
    }
```

These catch statements handle two error cases. The first is if the array goes out of bounds. The second is not as obvious, but it is there in case the programmer forgot initialize the array. Initially java arrays default to NULL until you use the new statement to allocate them memory.

This set of statements could also be rewritten in a slightly more generic way:

```
int i = 0;

try {
    for( i = 0; i < 9; i++ )
        myArray[i] = i;
}

catch ( RuntimeException e ) {
    System.out.println("Sorry, an error has occurred.");
    e.printStackTrace();
}
```

The RuntimeException e can catch any runtime exception, however since it is so generic you may have a difficult time diagnosing the problem. If you do not want to do extensive error checking, this is the easiest way (though now the safest way) to go.

Statements enclosed in the finally command at the end of a series of try and catch statements are guaranteed to run. This is extremely useful for doing data cleanup as a precaution or before the program crashes.

Suppose:

```
try
{
    code possibly with errors
}
catch
{ error handling code }
finally
{
    //Save Data
    //Cleanup
}
```

The key thing about the finally statement is that it is virtually guaranteed to always run no matter what. This is true whether you have exceptions or not, or if there are return statements in the try or catch blocks. One exception to this rule is if you make a direct call to `System.exit(0)` which will immediately crash your program.

Break and Continue

Break and continue are useful command for controlling the behavior of a Java program from within a loop. If a break command is received with a loop (such as a while loop) the program will immediately skip the rest of the loop and begin executing the command after the loop. Break is often used in conjunction with infinite loops:

```
while ( true )
{
    if ( x == 7 )
        break;

    additional code
}
System.out.println("The loop is done!");
```

At the start of the loop, if x happens to be 7, the loop is “broken” and “The loop is done!” is printed to the console because it is the next command after the loop.

Continue works similarly to break except that after skipping the rest of the loop it returns to the beginning of the loop instead of leaving the loop altogether:

```
while ( x != 9 )
{
    code

    if ( x == 8 )
        continue;

    additional code
}
```

With this code, anytime x equals 8, the additional code in the loop is skipped. The program then returns to the top of the loop and checks to see if x is equal to 9 and continues the loop as normal.

5. Key Differences between C++ and Java:

No pointers

In Java there are no pointers just references. This basically means there is no way for you to directly access the memory that has been reserved for an object except through the use of the object or primitive itself. It is possible to have multiple objects of the same type referencing the same memory.

In C++ it is possible to have multiple object and primitive pointers pointing at the same block of memory and for each to operate on it in a different way.

In Java, once the memory is reserved for a particular data type you can only operate on the memory it references with objects of that data type.

No delete

One of the most onerous and difficult tasks in C++ is keeping track of objects and then deleting them to free up memory and prevent data leaks. In Java there is no delete command because Java does memory garbage collection for free. There is a slight performance decrease associated with this service, but it greatly aids in providing reliable crash free programs.

Everything is object oriented

In C++ you can have functions that are not attached to classes. The main function in C++ is explicitly not attached to a class.

In Java, all function, including the main function, are attached to classes. The class with the main function in its implementation is the object in which the program originates. This object is automatically instantiated when the program executes.

Use of catch and try mandatory

For the most severe and common errors in Java the use of catch and try is mandatory. It is built into Java's grammar that these error checking precautions must be present or else your program will not compile.

In C++, catch and try are also available, but are not mandatory.

No Multiple Inheritance

In C++ one class can inherit from multiple super classes. In Java, to prevent complicated class trees, a class can only inherit from one superclass.

6. Java Compilation Basics

Check your Classpath Variable

Java and Javac Execution and Common Switches

Java invokes the JVM and runs your java program (class file)

Execution: `java <switches> <java program>`

Example: `java -verbose helloworld`

It is important to note is that when you run a java program using the java command you leave the .class extension off the file name. Look at the example listed above.

Execution: `javac <switches> <java program>`

Example: `javac -verbose helloworld.java`

Generates: `helloworld.class`

Javac is short for java compiler. It compiles your java code to the point where a JVM can finish the job. It is important to note that the output file's name will be the name of the class you compiled that contains the main function, not the name of the file. If you have a file named helloworld.java, but the class name inside helloworld.java with the main function is hWorld then my output file from javac is going to be hWorld.class.

Common switches for java and javac:

-help

Generates output that shows the template for running the command as well as a list of all the common switches.

-classpath

Specifies on the command line additional places java or javac should look for files necessary for your program to run correctly.

Example: `java -classpath /home/Students/lh1037/javafiles javaprog`

This specifies /home/Students/lh1037/javafiles as another place to look for files that the program requests.

-version

Displays the version of java or javac you are currently using. This is very important if you have multiple copies of java installed on your computer and you want to verify that the correct version is being used.

-verbose

Gives extra detail for all operations and problems that java or javac run into. This is most useful for javac as it will display all warnings and messages related to compilation.

To demonstrate java and javac we will compile and execute a few simple “hello world” programs whose code is listed below:

Hello World:

```
import java.io.*;

class helloWorld
{
    static public void main( String[] arguments)
    {
        System.out.println("Hello World!\n");
    }
}
```

Hello World with Swing:

```
import java.io.*;
import javax.swing.*;

public class helloSwing extends JFrame
{
    JButton hWorld = new JButton("Hello World!!!");

    public helloSwing()
    {
        super("Hello World!");
        setSize(300,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel topLayer = new JPanel();
        topLayer.add(hWorld);
        setContentPane(topLayer);
    }

    public static void main( String[] arguments)
```

```
{
    helloSwing hf = new helloSwing();
    hf.show();
}
}
```

7. Advanced Java Topics

Advanced topics are best handled with code examples. Several topics are covered in this section, but the two main topics are JDBC and File I/O.

File I/O is covered with the fortune cookie / greeting program below. Also covered is the use of inheritance, the super command, and a little more swing.

addMessage.java:

```
import java.io.*;
import java.util.Random;
import javax.swing.*;

//This base class is inherited by addMessage

class basicMessage
{
    //class variables

    int numMessages;
    String[] messages;

    //constructor

    public basicMessage()
    {
        numMessages = 0;
    }

    void outputMessages()
    {
        for( int i = 0; i < numMessages; i++)
            System.out.println(messages[i]);
    }

    void outputRandomGreeting()
    {
        System.out.println(messages[randomMessageIndex()]);
    }

    void getMessages()
    {
        try {
            FileReader file = new FileReader("greetings.txt");
            BufferedReader getMessages = new BufferedReader(file);

            String line = getMessages.readLine();

            numMessages = Integer.parseInt(line);
            messages = new String[numMessages];

            for( int i = 0; i < numMessages; i++)
```

```

        messages[i] = getMessages.readLine();

        getMessages.close();
    }
    catch(IOException e)
        { System.out.println("Error! : " + e.toString() ); }
}

int randomMessageIndex()
{
    if ( numMessages <= 0 )
        return 0;

    Random generator = new Random();
    return generator.nextInt(numMessages);
}
}

class addMessage extends basicMessage
{
    public addMessage()
    {
        super();
    }

    public static void main( String[] arguments)
    {
        addMessage greeter = new addMessage();
        greeter.getMessages();
        greeter.outputRandomGreeting();
        greeter.newFortune();
        System.exit(0);
    }

    void newFortune()
    {
        int response = JOptionPane.showConfirmDialog(null,
your own fortune?",
                                                    "Would you like to add
                                                    "Add a Fortune?",
JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE);
        if( response == 0 )
        {
            String newFortune = JOptionPane.showInputDialog(null,"Enter new
fortune:");
            addGreeting(newFortune);

```

```

    }
}

void addGreeting( String fortune)

{
    try {
        PrintWriter outputMessages = new PrintWriter(new
        FileWriter("greetings.txt"));

        outputMessages.println(numMessages+1);

        for( int i = 0; i < numMessages; i++)
            outputMessages.println(messages[i]);

        outputMessages.println(fortune);
        outputMessages.close();

    }
    catch(IOException e)
        { System.out.println("Error! : " + e.toString() ); }

}
}

```

greetings.txt (text file that holds greetings; each line is ended by the enter key)

```

6
Wise man says: "Take a cookie, leave a cookie"
Your dreams reveal important truths.
Do not desire what you do not need.
A modest man never talks to himself.
Treasure what you have.
The Lucky man never works early.

```

This program contains two classes, `addMessage` and `basicMessage`.

`basicMessage` opens `greetings.txt` and reads all the greetings into an array called `messages`. It then generates a random number and outputs the message at that index to the screen in a message box.

`addMessage` extends `basicMessage` via inheritance and adds the capability for the user to add their own custom fortune after reading their current fortune.

This program makes use of simple swing objects, file wrappers, the “super” keyword and basic try and catch structure.

Databases are covered below:

sentrydb.java:

```
import java.io.*;
import java.sql.*;

//username:lucas
//pass:9i8u7y6t
//dbname:lucas

public class sentrydb
{
    public static void main(String[] arguments)
    {
        String url
="jdbc:mysql://sentry.cs.txstate.edu/lucas?lucas&9i8u7y6t";
        String createTable = "create table PRODUCTS " +
            "(NAME varchar(32), " +
            " ID int, " +
            " PRICE float)";
        String dropTable = "drop table PRODUCTS";

        Connection con;
        Statement stmt;

        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(url,"lucas",
"9i8u7y6t");

            stmt = con.createStatement();

            stmt.executeUpdate(createTable);
            //stmt.executeUpdate(dropTable);

            stmt.close();
            con.close();

        }

        catch(java.lang.ClassNotFoundException e)
        {
            System.err.print("Error1: ");
            System.err.println(e.getMessage());
        }

        catch(java.lang.Exception e)
        {
            System.err.print("Error2: ");
            System.err.println(e.getMessage());
        }
    }
}
```

```
}  
}
```

This program connects to a mysql account on sentry.cs.txstate.edu and creates a table. It can also drop the same table if the code is modified slightly. This program covers the basics of forming a JDBC connection.

The proper JDBC driver must be installed on your system, in this case mysql's J/Connector driver, for the JDBC connection to work correctly.

8. Useful Links For Java:

Sun's Java Homepage:

<http://java.sun.com/>

Getting Java:

<http://java.sun.com/j2ee/1.4/download.html>

Sun's J2EE Tutorial:

<http://java.sun.com/j2ee/1.4/download.html#tutorial>

Free Java Text Book (Excellent Source)

<http://math.hws.edu/javanotes/index.html>

Installing Java:

<http://java.sun.com/j2se/1.5.0/install.html>

Sun's Documentation Page for J2EE

<http://java.sun.com/j2ee/1.4/docs/index.html>

Sun's JDBC Tutorial Page:

<http://java.sun.com/docs/books/tutorial/jdbc/basics/>

Mysql's J/Connector Driver Download Page

<http://dev.mysql.com/downloads/>

MySql's J/Connector Driver Documentation

<http://dev.mysql.com/doc/connector/j/en/index.html>

9. Code Appendix

Hello World:

```
import java.io.*;

class helloWorld
{
    static public void main( String[] arguments)
    {
        System.out.println("Hello World!\n");
    }
}
```

Hello World with Swing:

```
import java.io.*;
import javax.swing.*;

public class helloSwing extends JFrame
{
    JButton hWorld = new JButton("Hello World!!!");

    public helloSwing()
    {
        super("Hello World!");
        setSize(300,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel topLayer = new JPanel();
        topLayer.add(hWorld);
        setContentPane(topLayer);
    }

    public static void main( String[] arguments)
    {
        helloSwing hf = new helloSwing();
        hf.show();
    }
}
```

addMessage.java:

```
import java.io.*;
import java.util.Random;
import javax.swing.*;

//This base class is inherited by addMessage

class basicMessage
{
```

```

//class variables

int numMessages;
String[] messages;

//constructor

public basicMessage()
{
    numMessages = 0;
}

void outputMessages()
{
    for( int i = 0; i < numMessages; i++)
        System.out.println(messages[i]);
}

void outputRandomGreeting()
{
    System.out.println(messages[randomMessageIndex()]);
}

void getMessages()
{
    try {
        FileReader file = new FileReader("greetings.txt");
        BufferedReader getMessages = new BufferedReader(file);

        String line = getMessages.readLine();

        numMessages = Integer.parseInt(line);
        messages = new String[numMessages];

        for( int i = 0; i < numMessages; i++)
            messages[i] = getMessages.readLine();

        getMessages.close();
    }
    catch(IOException e)
        { System.out.println("Error! : " + e.toString() ); }
}

int randomMessageIndex()
{
    if ( numMessages <= 0 )
        return 0;

    Random generator = new Random();
    return generator.nextInt(numMessages);
}

```

```

}

class addMessage extends basicMessage
{
    public addMessage()
    {
        super();
    }

    public static void main( String[] arguments)
    {
        addMessage greeter = new addMessage();
        greeter.getMessages();
        greeter.outputRandomGreeting();
        greeter.newFortune();
        System.exit(0);
    }

    void newFortune()
    {
        int response = JOptionPane.showConfirmDialog(null,
                                                    "Would you like to add
your own fortune?",
                                                    "Add a Fortune?",
                                                    JOptionPane.YES_NO_OPTION,
                                                    JOptionPane.QUESTION_MESSAGE);
        if( response == 0 )
        {
            String newFortune = JOptionPane.showInputDialog(null,"Enter new
fortune:");
            addGreeting(newFortune);
        }
    }

    void addGreeting( String fortune)
    {
        try {
            PrintWriter outputMessages = new PrintWriter(new
FileWriter("greetings.txt"));

            outputMessages.println(numMessages+1);

            for( int i = 0; i < numMessages; i++)
                outputMessages.println(messages[i]);

            outputMessages.println(fortune);
            outputMessages.close();
        }
    }
}

```

```

    catch(IOException e)
        { System.out.println("Error! : " + e.toString() ); }

    }
}

```

greetings.txt (text file that holds greetings; each line is ended by the enter key)

```

6
Wise man says: "Take a cookie, leave a cookie"
Your dreams reveal important truths.
Do not desire what you do not need.
A modest man never talks to himself.
Treasure what you have.
The Lucky man never works early.

```

sentrydb.java:

```

import java.io.*;
import java.sql.*;

//username:lucas
//pass:9i8u7y6t
//dbname:lucas

public class sentrydb
{
    public static void main(String[] arguments)
    {
        String url
="jdbc:mysql://sentry.cs.txstate.edu/lucas?lucas&9i8u7y6t";
        String createTable = "create table PRODUCTS " +
            "(NAME varchar(32), " +
            " ID int, " +
            " PRICE float)";
        String dropTable = "drop table PRODUCTS";

        Connection con;
        Statement stmt;

        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            con = DriverManager.getConnection(url,"lucas",
"9i8u7y6t");

            stmt = con.createStatement();

            stmt.executeUpdate(createTable);
            //stmt.executeUpdate(dropTable);

```

```
        stmt.close();
        con.close();
    }

    catch(java.lang.ClassNotFoundException e)
    {
        System.err.print("Error1: ");
        System.err.println(e.getMessage());
    }

    catch(java.lang.Exception e)
    {
        System.err.print("Error2: ");
        System.err.println(e.getMessage());
    }
}
}
```