Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

SAIRAM DIGITAL RESOURCES

YEAR II SEM III

CS8392

OBJECT ORIENTED PROGRAMMING
(Common to CSE, EEE, EIE, ICE, IT)

UNIT NO 3

Exception Handling and I/O

3.5 Stack Trace Elements

COMPUTER SCIENCE & ENGINEERING

Stack Trace elements

- StackTraceElement class **element** represents a single **stack** frame. All **stack** frames except for the one at the top of the **stack** represent a method invocation.The frame at the top of the **stack** represents the execution point at which the **stack trace** was generated.

- The getStackTrace() method of thread class returns an array of stack trace elements representing the stack dump of the thread. The first element of an array represents the top of the stack which is the last method invocation in the sequence. The last element of the array represents the bottom of the stack which is the first method invocation in the sequence.

Syntax:

```
public StackTraceElement[] getStackTrace()
```

Return

It is an array of StackTraceElement, each represents one stack frame.

- All stack frames except for the one at the top of the stack represent a method invocation.The frame at the top of the stack represent the execution point of which the stack trace was generated.
- Each stack frame represents an execution point, which includes such things as the name of the method, the name of file and the source code line number.
- An array of **StackTraceElement** is returned by **getStackTrace()** method of the**Throwable** class.

Constructor: Creates a stack trace element representing the specified execution point.

```
StackTraceElement(String declaringClass, String methodName, String fileName, int lineNumber)
```

Parameters:

declaringClass – the fully qualified name of the class containing the execution point represented by the stack trace element.

methodName – the name of the method containing the execution point represented by the stack trace element.

fileName – the name of the file containing the execution point represented by the stack trace element, or null if this information is unavailable

lineNumber – the line number of the source line containing the execution point represented by this stack trace element, or a negative number if this information is unavailable. A value of -2 indicates that the method containing the execution point is a native method.

Throws: NullPointerException – if declaringClass or methodName is null.

METHODS

boolean equals(ob): Returns try if the invoking **StackTraceElement** is as the one passed in **ob**. Otherwise it returns false.

Syntax: public boolean equals(ob)
Returns: true if the specified object is another StackTraceElement instance representing the same execution point as this instance.
Exception: NA

```
import java.lang.*;
import java.io.*;
import java.util.*;
public class StackTraceElementDemo
{
   public static void main(String[] arg)
   {
      StackTraceElement st1 = new StackTraceElement("foo", "fuction1",  "StackTrace.java", 1);
      StackTraceElement st2 = new StackTraceElement("bar", "function2", "StackTrace.java", 1);

      Object ob = st1.getFileName();

      // checking whether file names are same or not
      System.out.println(st2.getFileName().equals(ob));
   }
}
```

OUTPUT:
TRUE

String getClassName(): Returns the class name of the execution point described by the invoking **StackTraceElement**.

Syntax: public String getClassName().
Returns: the fully qualified name of the Class containing the execution point represented by this stack trace element.
Exception: NA.

```
import java.lang.*;
import java.io.*;
import java.util.*;
public class StackTraceElementDemo
{
   public static void main(String[] arg)
   {
      System.out.println("Class name of each thread involved:");
      for(int i = 0; i<2; i++)
      {
         System.out.println(Thread.currentThread().getStackTrace()[I].
         getClassName());
      }
   }
}
```

OUTPUT:
Class name of each thread involved:
java.lang.Thread
StackTraceElementDemo

String getFileName(): Returns the file name of the execution point described by the invoking **StackTraceElement**.

Syntax: public String getFileName().
Returns: the name of the file containing the execution point represented by this stack trace element,or null if this information is unavailable.
Exception: NA.

```
import java.lang.*;
import java.io.*;
import java.util.*;
public class StackTraceElementDemo
{
   public static void main(String[] arg)
   {
       System.out.println("file name: ");
       for(int i = 0; i<2; i++)
       System.out.println(Thread.currentThread().getStackTrace()[i].
        getFileName());
   }
}
```

OUTPUT:

```
Thread.java
StackTraceElementDemo.java
```

int getLineNumber(): Returns the source-code line number of the execution point described by the invoking **StackTraceElement**. In some situation the line number will not be available, in which case a negative value is returned.

Syntax: public int getLineNumber().
Returns: the line number of the source line containing the execution point represented by this stack trace element, or a negative number if this information is unavailable.
Exception: NA.

```
import java.lang.*;
import java.io.*;
import java.util.*;
public class StackTraceElement Demo
{
  public static void main(String[] arg)
  {
     System.out.println("line number: ");
     for(int i = 0; i<2; i++)
     System.out.println(Thread.currentThread().getStackTrace()[i].
      getLineNumber());
  }
}
```

Output:

```
line number:
1556
10
```

String getMethodName(): Returns the method name of the execution point described by the invoking **StackTraceElement**.

Syntax: public String getMethodName().
Returns: the name of the method containing the execution point represented by this stack trace element.
Exception: NA.

```
import java.lang.*;
import java.io.*;
import java.util.*;
public class StackTraceElementDemo
{
   public static void main(String[] arg)
   {
      System.out.println("method name: ");
      for(int i = 0; i<2; i++)
      System.out.println(Thread.currentThread().getStackTrace()[i].
       getMethodName());
   }
}
```

OUTPUT

```
method name:
getStackTrace
main
```

int hashCode(): Returns the hash code of the invoking **StackTraceElement**.

Syntax: public int hashCode().
Returns: a hash code value for this object.
Exception: NA.

```
import java.lang.*;
import java.io.*;
import java.util.*;
public class StackTraceElementDemo
{
   public static void main(String[] arg)
   {
      System.out.println("hash code: ");
      for(int i = 0; i<2; i++)
      System.out.println(Thread.currentThread().getStackTrace()[i].
       hashCode());
   }
}
```

OUTPUT

```
hash code:
-1225537245
-1314176653
```

boolean isNativeMethod(): Returns true if the invoking **StackTraceElement** describes a native method. Otherwise returns false.

Syntax: public boolean isNativeMethod().
Returns: true if the method containing the execution point represented by this stack trace element is a native method.
Exception: NA.

```
import java.lang.*;
import java.io.*;
import java.util.*;
public class StackTraceElementDemo
{
   public static void main(String[] arg)
   {

      for(int i = 0; i<2; i++)
      System.out.println(Thread.currentThread().getStackTrace()[i].
       isNativeMethod());
   }
}
```

OUTPUT

```
false
false
```

String toString(): Returns the String equivalent of the invoking sequence.

Syntax: public String toString().
Returns: a string representation of the object.
Exception: NA.

.

```
import java.lang.*;
import java.io.*;
import java.util.*;
public class StackTraceElementDemo
{
   public static void main(String[] arg)
   {
       System.out.println("String equivlaent: ");
       for(int i = 0; i<2; i++)
       System.out.println(Thread.currentThread().getStackTrace()[i].
        toString());
   }
}
```

OUTPUT

```
String equivlaent:
java.lang.Thread.getStackTrace
StackTraceElementDemo.main
```