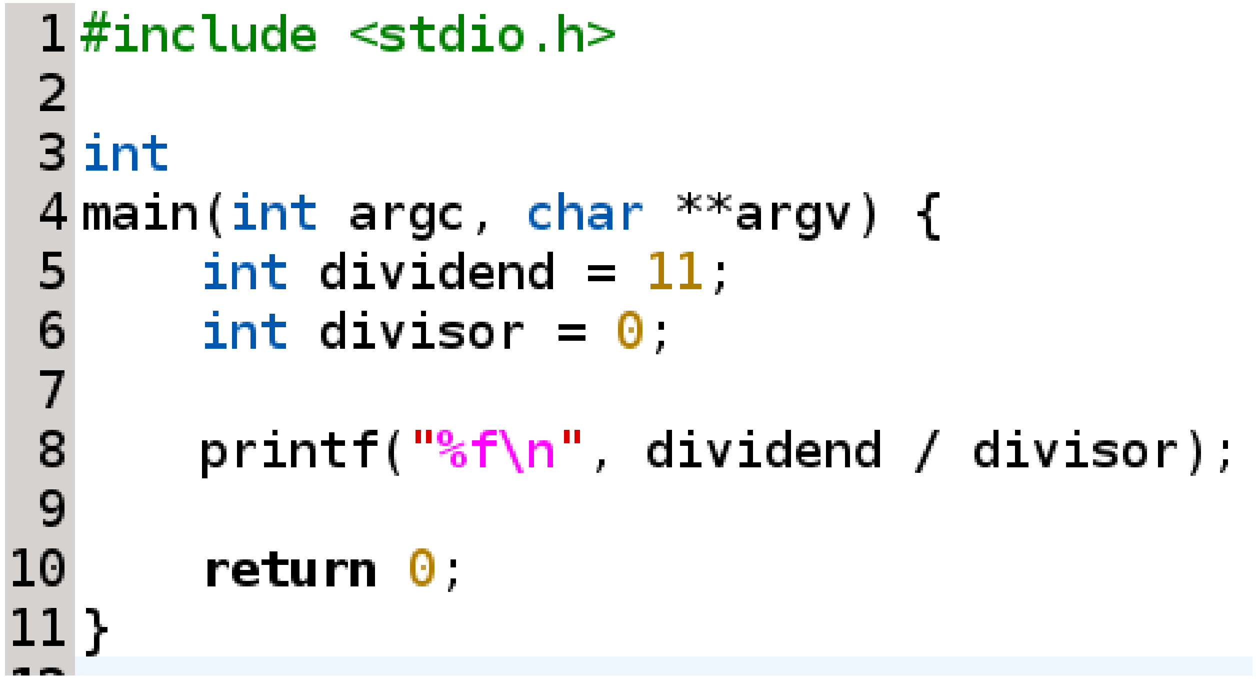
Java: Exception Handling and String Handling

Professor Sudip Misra
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur
http://cse.iitkgp.ac.in/~smisra/

Introduction

- Consider the following C program
  - What would be its output?



```c
#include <stdio.h>

int
main(int argc, char **argv) {
    int dividend = 11;
    int divisor = 0;

    printf("%f\n", dividend / divisor);

    return 0;
}
```

Introduction (Cont'd)

- On execution, it shows “Divide Error” (using TC++ 3.2)
- Let us fix it:

```c
#include <stdio.h>

int
main(int argc, char **argv) {
    int dividend = 11;
    int divisor = 0;

    if (divisor != 0) {
        printf("%f\n", dividend / divisor);
    } else {
        printf("Result undefined!\n");
    }

    return 0;
}
```

What is an Exception?

- A run-time error
- An abnormal condition that arises during execution of a program
  - May or may not occur depending upon the situation
- In contrast, compilation errors
  - Arise during compilation time (e.g., due to wrong syntax)
  - Unless fixed, compilation would keep failing
- One should always handle all possible exceptions
  - Otherwise, the program may crash during execution

Exception Handling in Java

- Exception object: Represents a particular type of exception
- When abnormality arises during execution of a given statement of code, an exception is thrown
- Methods throwing exceptions are indicated by **`throws`**
- Also, you can **`throw`** exceptions manually
- All thrown exceptions are caught (by you or JVM; it's better you **`try`** and **`catch`** them)
- If there is anything that must be executed at the end of try (irrespective of whether or not there was an exception), do it **`finally`**`(optinal)`
- Five keywords: try, catch, throw, throws, finally

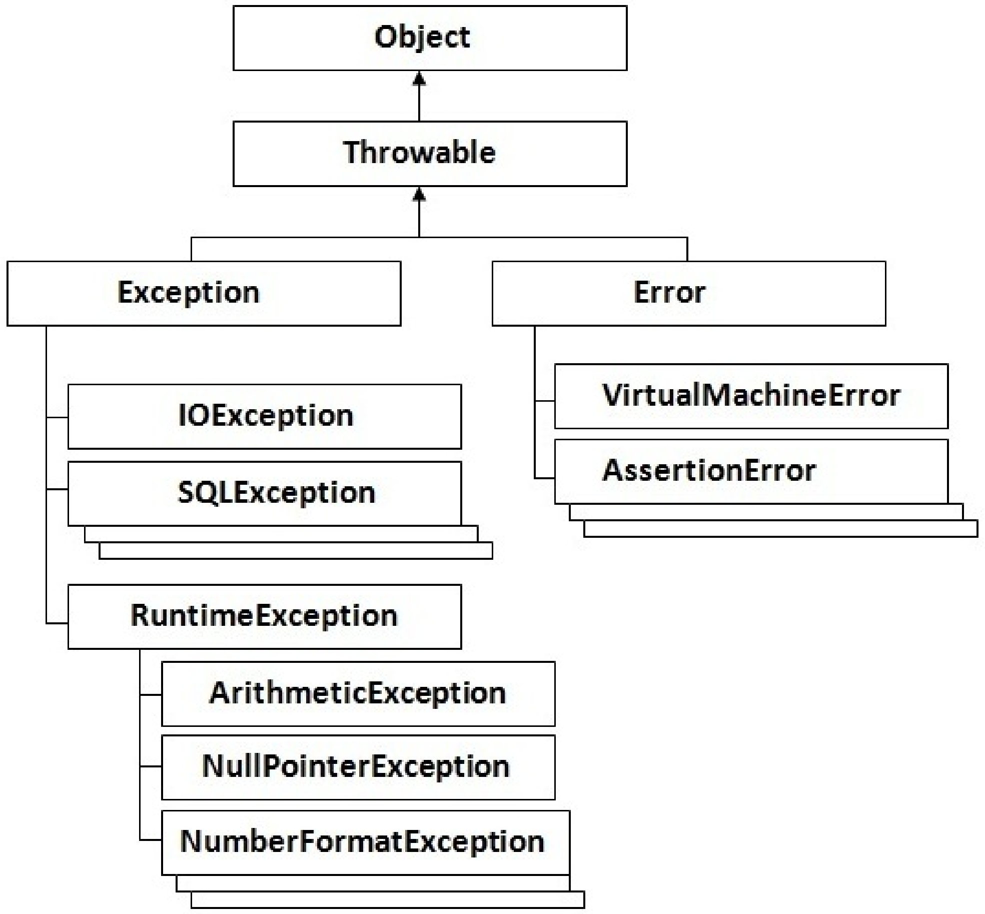
TODO: Learn the difference among `final`, `finally`, and `finalize` in Java

Exception Handling in Java (Cont'd)

- Let us say that a code sequence throws two exceptions

```
try {
   Statement1;
   Statement2;
} catch (Exception1 ex1) {
   // Handle Exception1
} catch (Exception2 ex2) {
   // Handle Exception2
} finally {
    // Optional Block
   // Code that must be executed before the try block ends
   // Cleanup code can be put here (e.g., close database
     connection)
   // Java 7 introduced try-with-resources where this is
     not required anymore
}
```

EXCEPTIONS HIERARCHY [1]



[1]: http://www.javatpoint.com/exception-handling-in-java

Types of Exceptions

- Unchecked:
  - Not checked at compile-time, but at run-time
  - "Automatically" thrown even if there is no try/throw
  - Subclasses of the `Error` and `RuntimeException` classes
- Examples:
  - `NullPointerException`: Object reference points to null
  - `ArrayIndexOutOfBoundsException`: Incorrect array index
  - `AssertionError`: An assertion has failed
    - An assertion is a boolean condition upon a set of variables/methods
    - Program execution stops if an assertion fails
    - Assertions are widely used for testing purposes
    - Special flag (-ea) must be passed to the Java interpreter to enable assertions

Types of Exceptions (Cont'd)

- Checked:
  - Checked at compile-time
  - If a code block within a method throws any checked exception, then either
    - There should be a try/catch block, or
    - The method should use **throws**, and throw the exception(s)
  - Subclasses of `Exception` class excluding the `RuntimeException` class
- Examples:
  - `IOException`: Exceptions related to I/O access
  - `SQLException`: Examples related to database query

Types of Exceptions (Cont'd)

- Error:
  - Indicates some serious problem encountered during program execution
  - Subclass of the Error class
- Examples:
  - `IOError`: Error during I/O access
  - `VirtualMachineError`: JVM has probably run out of resources
- See [1] for examples on code resulting in different types of exceptions

Exceptions within Exceptions within …

```java
public class NestedExceptionExample {
    public static void main(String[] args) {
        int result = -1;

        try {
            System.out.println("Let us divide pie among zero people");
            result = 22 / 7 / 0;
        } catch (ArithmeticException ae) {
            System.out.println("Oh! An exception occurred! " + ae);

            // We are still stubborn to divide
            try {
                int[] pies = new int[0];
                pies[0] = 0;
            } catch (ArrayIndexOutOfBoundsException aie) {
                System.out.println("Another exception! " + aie);
                // We now give up
                result = 0;
            } // End of ae try
        } // End of ae try

        System.out.println("Everybody gets " + result + " pies");
    }
}
```

Execution Order

- Line # 7 triggers `ArithmeticException`
- Control goes to the catch block in line # 8
  - Line # 14 refers wrong array index
  - Triggers `ArrayIndexOutOfBoundsException`
  - Control goes to catch block in line # 15
    - Assigns result to 0
  - The inner catch block is done
- The outer catch block is done
- Prints in line # 22

Execution Order (Cont'd)

- Each try block must be accompanied with at least one catch block
- The scope of a catch block(s) is(are) limited only to its(their) immediately preceding try block
- Exception objects (`ae` and `aie` in the example) provide description of the concerned exception
  - We can print them as strings
  - Other useful info also available, e.g., stack trace

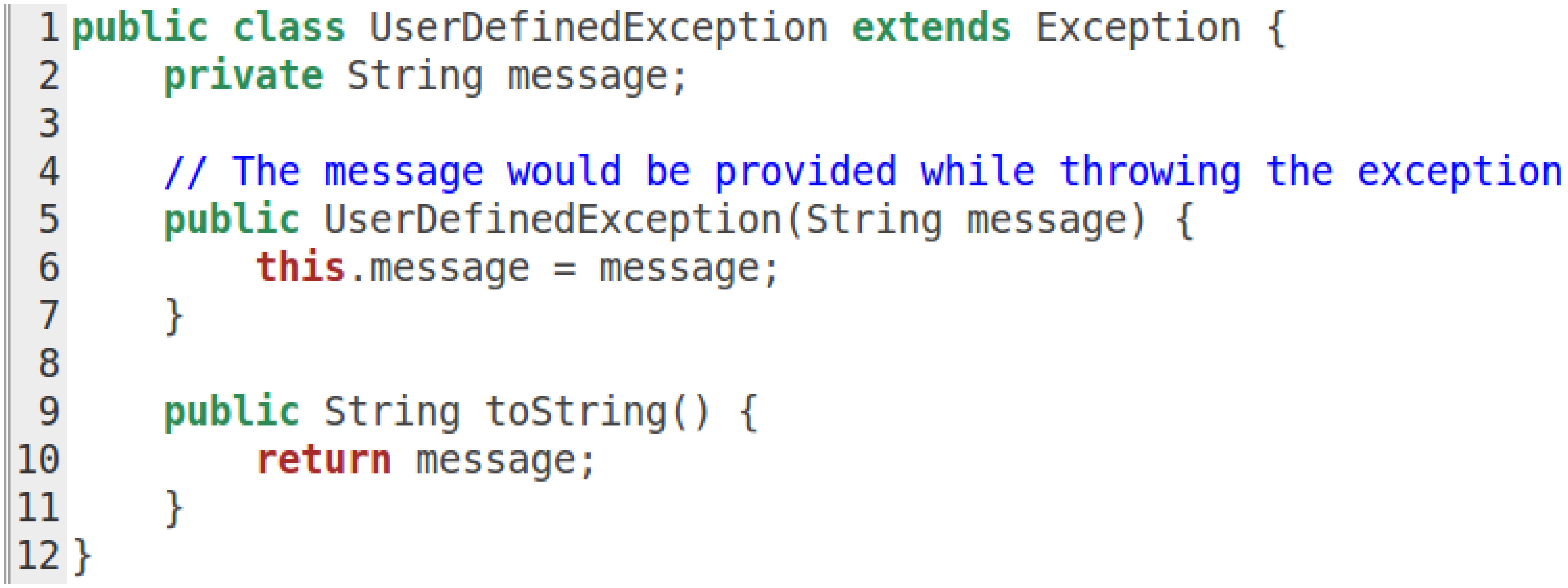
Output

```
Let us divide pie among zero people
Oh! An exception occurred!
  java.lang.ArithmeticException: / by zero
Another exception!
  java.lang.ArrayIndexOutOfBoundsException:
  0
Everybody gets 0 pies
```

User Defined Exceptions

- How to define your own exception?
  - Create a subclass of the
    - `Exception` class for checked exceptions, or
    - `RuntimeException` class for unchecked exceptions
  - Provide a constructor [optional]
  - Override the `toString()` method to provide customized description, if relevant
    - The `toString()` method returns the string/textual representation of any object
- How to use it?
  - throw from your code

THE EXCEPTION CLASS

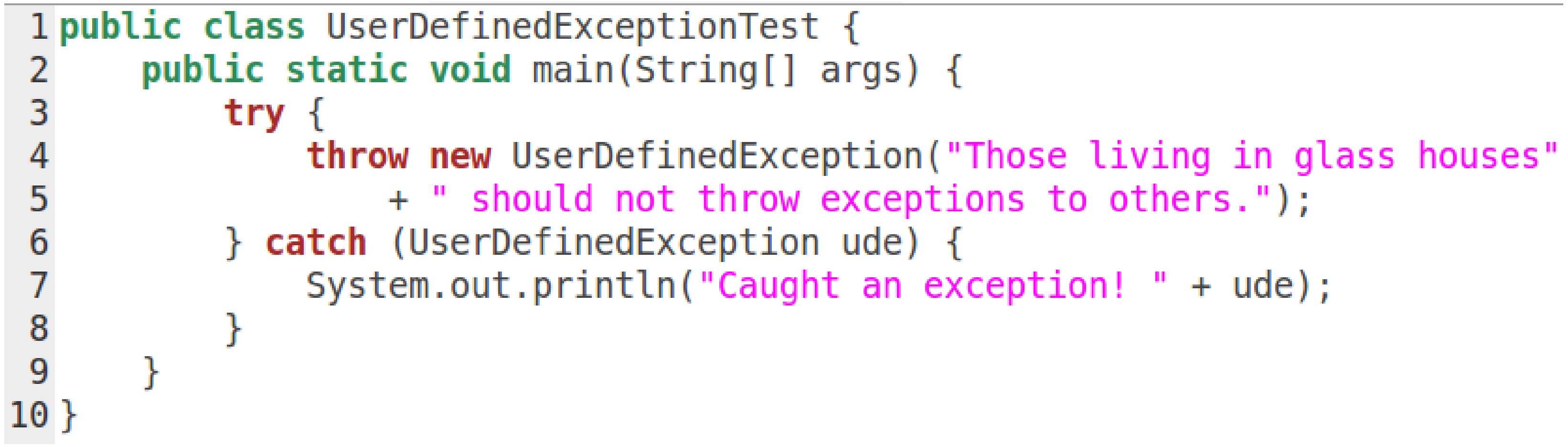


```java
public class UserDefinedException extends Exception {
    private String message;

    // The message would be provided while throwing the exception
    public UserDefinedException(String message) {
        this.message = message;
    }

    public String toString() {
        return message;
    }
}
```

Throwing the Exception

- Line # 4 throws the exception
- Line # 6 catches it; line # 7 prints the custom message
- Output:

```
Caught an exception! Those living in glass
  houses should not throw exceptions to others.
```



```
 1 public class UserDefinedExceptionTest {
 2     public static void main(String[] args) {
 3         try {
 4             throw new UserDefinedException("Those living in glass houses"
 5                 + " should not throw exceptions to others.");
 6         } catch (UserDefinedException ude) {
 7             System.out.println("Caught an exception! " + ude);
 8         }
 9     }
10 }
```

Throw Without Exception Handling

```java
public class UserDefinedExceptionTest {
    public static void main(String[] args) {
        try {
            throw new UserDefinedException("Those living in glass houses"
                + " should not throw exceptions to others.");
        } catch (UserDefinedException ude) {
            System.out.println("Caught an exception! " + ude);
        }

        // Invoke a method that throws exception
        try {
            exceptionThrower();
        } catch (UserDefinedException ude) {
            System.out.println("Caught an exception! " + ude);
        }
    }

    public static void exceptionThrower() throws UserDefinedException {
        throw new UserDefinedException("Yet they do.");
    }
}
```

Output

```
Caught an exception! Those living in glass
  houses should not throw exceptions to
  others.
Caught an exception! Yet they do.
```

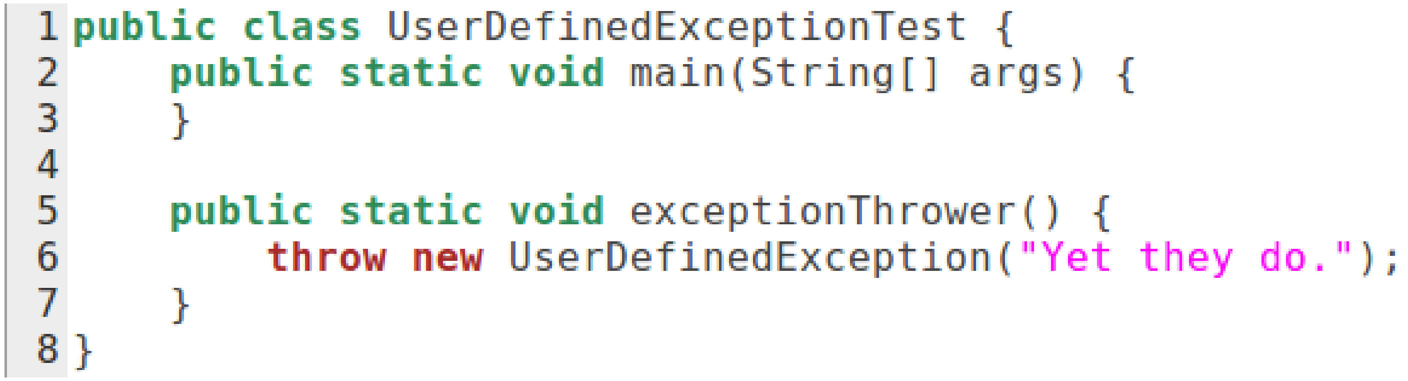
To throw or throws?

- throw:
  - The keyword is used inside methods to explicitly throw an exception
  - A single throw statement can trigger only a single exception
- throws:
  - Any method that causes exception but does not catch, must declare them using throws
  - throws can list several uncaught exceptions, e.g.,

```
public static void exceptionThrower() throws
  UserDefinedException, ArithmeticException {
```

Is it throw or throws? (Cont'd)

- What would happen if there was no throws?



```java
public class UserDefinedExceptionTest {
    public static void main(String[] args) {
    }

    public static void exceptionThrower() {
        throw new UserDefinedException("Yet they do.");
    }
}
```

```
UserDefinedExceptionTest.java:6: unreported
  exception UserDefinedException; must be caught or
  declared to be thrown
        throw new UserDefinedException("Yet they
  do.");
        ^
1 error
```

Exception Propagation

- Exception occurring at the top of the stack propagates downward until a method is found that handles the exception
- Rule holds for unchecked exceptions
- Compilation error in case of checked exceptions

EXCEPTION PROPAGATION: EXAMPLE

```java
public class ExceptionPropagation {
    public static void main(String[] args) {
        try {
            divide(11, 0);
        } catch (ArithmeticException ae) {
            System.out.println("" + ae);
        }
    }

    public static int divide(int x, int y) {
        return division(x, y);
    }

    private static int division(int x, int y) {
        return x / y;
    }
}
```

Output & Explanation

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

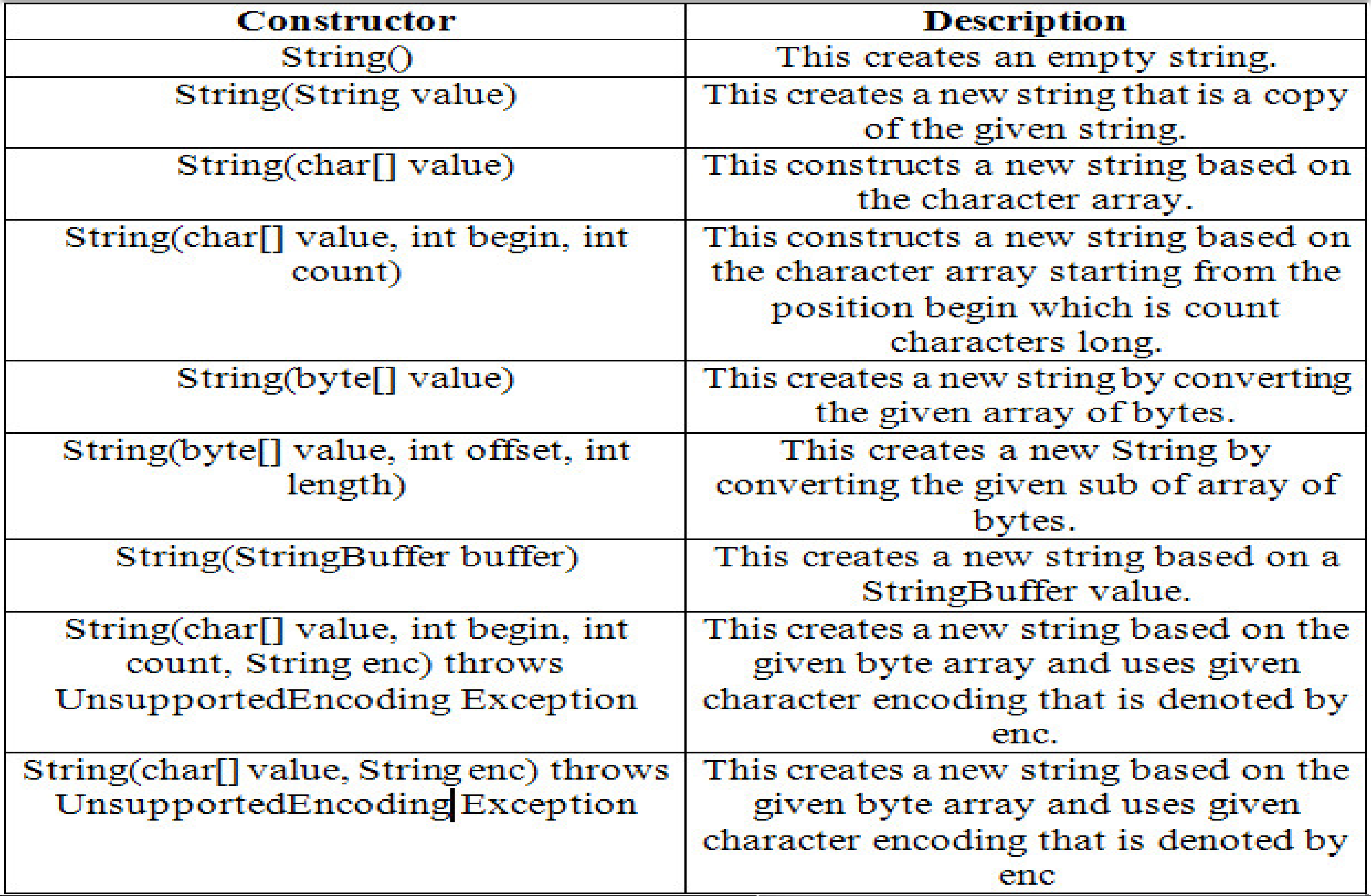
- Method call sequence:
  - `main()`
    - `divide()`
      - `division() // Triggers exception`
- Exception propagation (call stack; top to bottom):
  - `division() // No try-catch; go downward`
    - `divide() // No try-catch; go downward`
      - `main() // Has try-catch; handles the exception`

String Handling: String

- A sequence of characters.
- In Java, strings are treated as objects.
- The **java.lang.String** class is used to create string object.
- **Example**

  String s = new String(); //Creates an empty string

The String Constructors



Constructor	Description
String()	This creates an empty string.
String(String value)	This creates a new string that is a copy of the given string.
String(char[] value)	This constructs a new string based on the character array.
String(char[] value, int begin, int count)	This constructs a new string based on the character array starting from the position begin which is count characters long.
String(byte[] value)	This creates a new string by converting the given array of bytes.
String(byte[] value, int offset, int length)	This creates a new String by converting the given sub of array of bytes.
String(StringBuffer buffer)	This creates a new string based on a StringBuffer value.
String(char[] value, int begin, int count, String enc) throws UnsupportedEncoding Exception	This creates a new string based on the given byte array and uses given character encoding that is denoted by enc.
String(char[] value, String enc) throws UnsupportedEncoding Exception	This creates a new string based on the given byte array and uses given character encoding that is denoted by enc

String Length

- The length of a string is the number of characters that it contains.
- **length**() method returns the number of characters contained in the string object.

```
public class StringDemo {
  public static void main(String args[]) {
    String venue= new String(“Netaji Auditorium”);
    int len = venue.length();
    System.out.println( “String Length is : “ + len );
  }
}
```

Output:

```
String Length is : 17
```

Special String Operations : String Concatenation

- **concat** method

```
String s1 = new String("Welcome to ");
String s2 = "Java";
String s3 = s1.concat(s2);
System.out.println(s3);
```

Output:

```
s3 = "Welcome to Java"
```

- **plus (+)** sign

```
String s1 = new String("Welcome to ");
String s2 = "Java";
String s3 = s1 + s2;
System.out.println(s3);
```

Output:

```
s3 = "Welcome to Java"
```

The toString() method

- All classes that represent objects should define a **toString** method.
- The **toString** method returns a character string that represents the object in some way.
- It is called automatically when an object is concatenated to a string or when it is passed to the **print/println** method.

```
public String toString()
```

Character Extraction : charAt()

- Extracts a single character from a String.

  char charAt(int *loc*)

- **Example**

```
char ch;
ch = "abc".charAt(1);
```

  Assigns the value “b” to ch.

Character Extraction : getChars()

- Copies characters from this string into the destination character array.

public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)

where,

srcBegin – index of the first character in the string to copy.

srcEnd – index after the last character in the string to copy.

dst – the destination array.

dstBegin – the start offset in the destination array.

- **Example**

```
String Str1 = new String("Welcome to Java");
 char[] Str2 = new char[7];
 Str1.getChars(2, 9, Str2, 0);;
```

Copies the characters “lcome t” to Str2.

Character Extraction : getBytes()

- This method encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array.

public byte[] getBytes()

Character Extraction : toCharArray()

- Converts all the characters in a String object to a character array.

char[] toCharArray()

String Comparison : equals() and equalsIgnoreCase()

boolean equals(Object str)

boolean equalsIgnoreCase(String str)

Example :

“xyz”.equals(“abc”); // false

“xyz”.equalsIgnoreCase(“XYZ”); //true

s1.equals(s2); // true if s1 and s2 are equal

s1.equalsIgnoreCase(s2); //true if s1 and s2 are equal by ignoring case

String Comparison : startsWith() and endsWith()

boolean startsWith(String str)

boolean endsWith(String str)

- Used to check whether a string starts/ends with a string str or not.

Example :

“object”.startsWith(“obj”); // true

“Sachin plays cricket”.endsWith(“cricket”); //true

- Second Form of startsWith allows to specify the starting point:

boolean startsWith(String str, int startIndex);

“Sachin plays cricket”.startsWith(“plays”,7); // true

- endsWith has only one form

String Comparison : compareTo()

int compareTo(String str)

int compareToIgnoreCase(String str)

- Used for String comparisons.
- Returns one of the three possible values:

Value	Meaning
Less than zero	The invoking string is less than *str*.
Greater than zero	The invoking string is greater than *str*.
Zero	The two strings are equal.

- Used for ordering/sorting strings

SEARCHING STRINGS : INDEXOF() AND LASTINDEXOF()

- Used to search first/last occurrences of a character / substring.
- Return the index of character or substring if found, otherwise -1.
- These two methods are overloaded in several different ways :
  - **int indexOf(int ch)**
    int lastIndexOf(int ch)
  - **int indexOf(String str)**
    int lastIndexOf(String str)
  - **int indexOf(int ch, int startIndex)**
    int lastIndexOf(int ch, int startIndex)
  - **int indexOf(String str,startIndex) /**
    int lastIndexOf(String str, int startIndex)

Modifying a String : substring()

- Because String objects are immutable, whenever we want to modify a String, it will construct a new copy of the string with modifications.
- **substring**() method is used to extract a part of a string.

public String substring (int start_index)

public String substring (int start_index, int end_index)

Example:

```
            String s = "ABCDEFG";
            String t = s.substring(2);
            String u = s.substring (1, 4);
```

Substring t contains "CDEFG"

Substring u contains "BCD"

Note: Substring from start_index to end_index-1 will be returned.

MODIFYING A STRING : REPLACE()

The **replace**( ) method has two forms.

- The first replaces all occurrences of one character in the invoking string with another character. It has the following general form:

String replace(char original_char, char replacement)

Here, original_char specifies the character to be replaced by the character specified by replacement.

Example: String s = "Hello".replace('l', 'w');

- The second form of replace( ) replaces one character sequence with another. It has this general form:

String replace(CharSequence original, CharSequence replacement)

Modifying a String : trim()

- The **trim**( ) method returns a copy of the invoking string from which any leading and trailing whitespace has been removed.

String trim( )

- **Example**:

```
String s = " Hello World ".trim();
```

This puts the string "Hello World" into s.

Data Conversion Using valueOf()

String.valueOf(X)

- Returns String representation of X
- X: char, int, char array, double, float, Object
- Useful for converting different data types into String.
- **Example**

String str1 = String.valueOf(4); //returns “4”

String str2 = String.valueOf(‘A’); //returns “A”

String str3 = String.valueOf(40.02); //returns “40.02”

Changing the Case of Characters Within a String

- **toLowerCase**() - converts all the characters in a string from uppercase to lowercase.

  String toLowerCase()

- **toUpperCase**() - converts all the characters in a string from lowercase to uppercase.

  String toUpperCase()

Thank you!