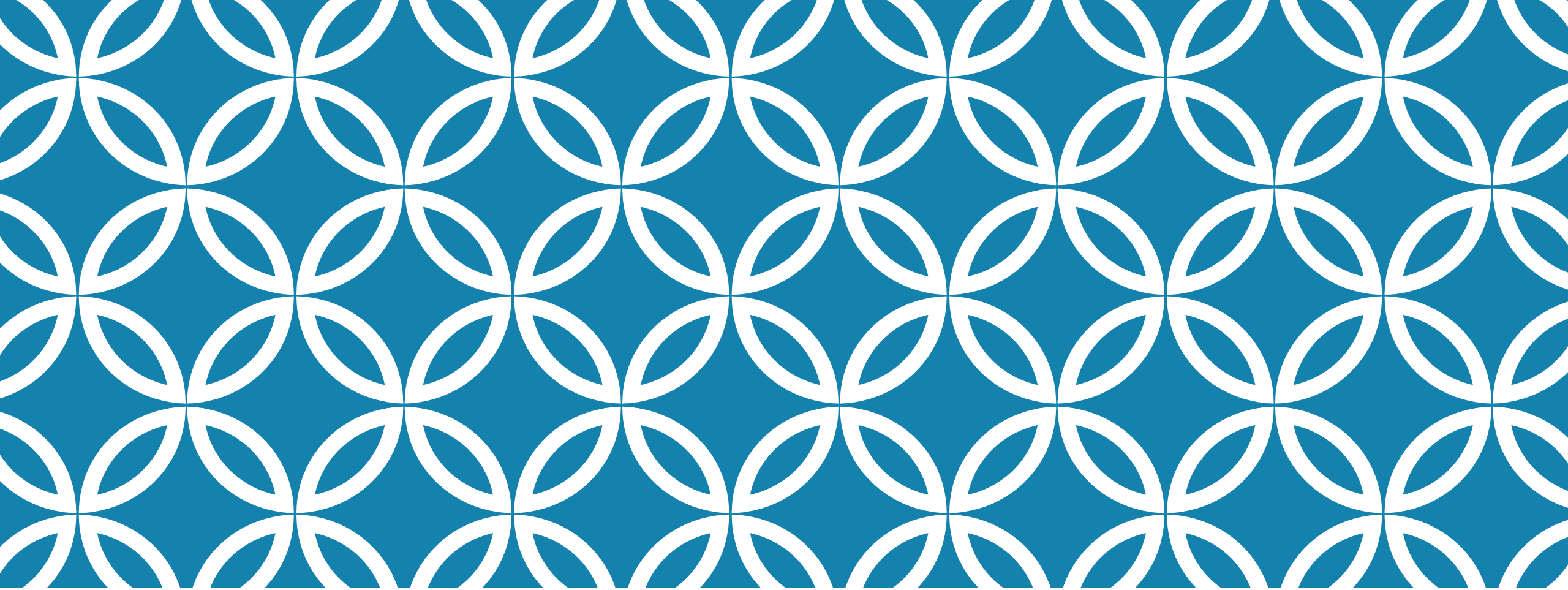


EXCEPTION HANDLING

Prof. Anita Agrawal,
BITS Pilani, K.K. Birla Goa campus

EXCEPTION HANDLING: AN INTRODUCTION

- An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time

- When an **Exception** occurs, the normal flow of the program is disrupted and the program/Application terminates abnormally,
  - This is not recommended, therefore, these exceptions are to be handled.

WHEN CAN AN EXCEPTION OCCUR???

- Some of the errors are caused by users, some by programmers, others by physical resources that have failed in some manner
- The user has entered invalid data
- A file that needs to be opened cannot be found
- A network connection has been lost in the middle of communication.
- The JVM has run out of memory

TYPES OF EXCEPTIONS

- Checked Exceptions (Compile time Exceptions):
  - An exception that is checked (notified) by the compiler at compilation-time. These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.

- Unchecked Exceptions (Run-Time Exceptions):
  - An exception that occurs at run-time.
  - Runtime exceptions are ignored at the time of compilation

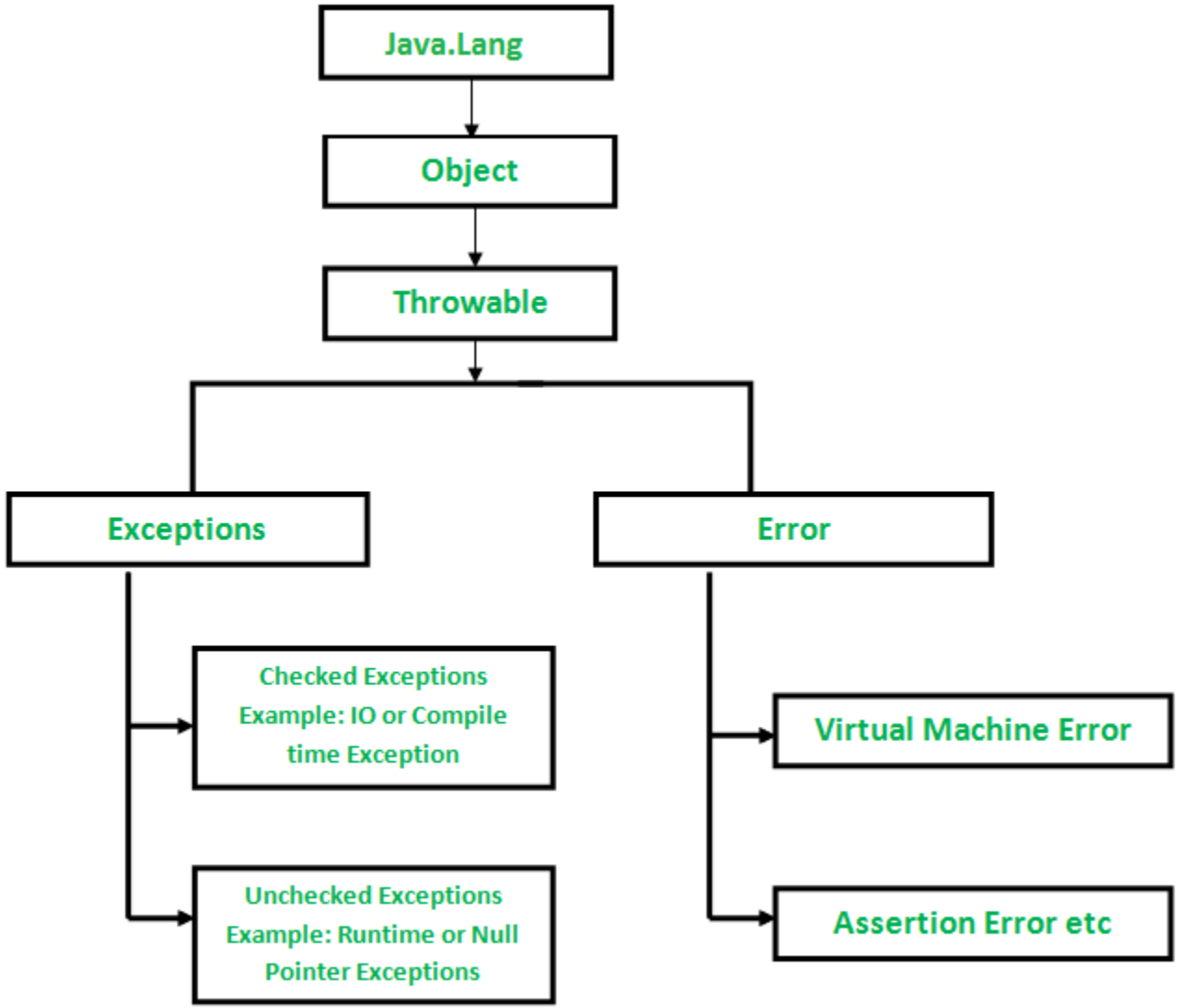
Examples: checked
unchecked

- **Not all computer languages support exception handling**
- Java supports Exception handling.
- It does so by bringing run-time error management into the object-oriented world.
- In Java, Exception is an object.
  - Describes an exceptional condition occurred in a piece of code

EXCEPTION HIERARCHY

- All exception classes are subtypes of the java.lang.Exception class
- The exception class is a subclass of the Throwable class.
- Throwable class also has another subclass called Error.

EXCEPTION CLASS



ERRORS

- Class Error defines the exceptions that are not expected to be caught by your program
- Are abnormal conditions that happen in case of severe failures such as catostrophic failures.
- These are not handled by the Java programs.
- **Errors** are used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE).
  - Example StackOverflow Error .

We will not be dealing with exceptions of type **Error**

ERROR TYPES

- Syntax errors arise because the rules of the language have not been followed. They are detected by the compiler
- Runtime errors occur while the program is running if the environment detects an operation that is impossible to carry out
- Logic errors occur when a program doesn't perform the way it was intended to.

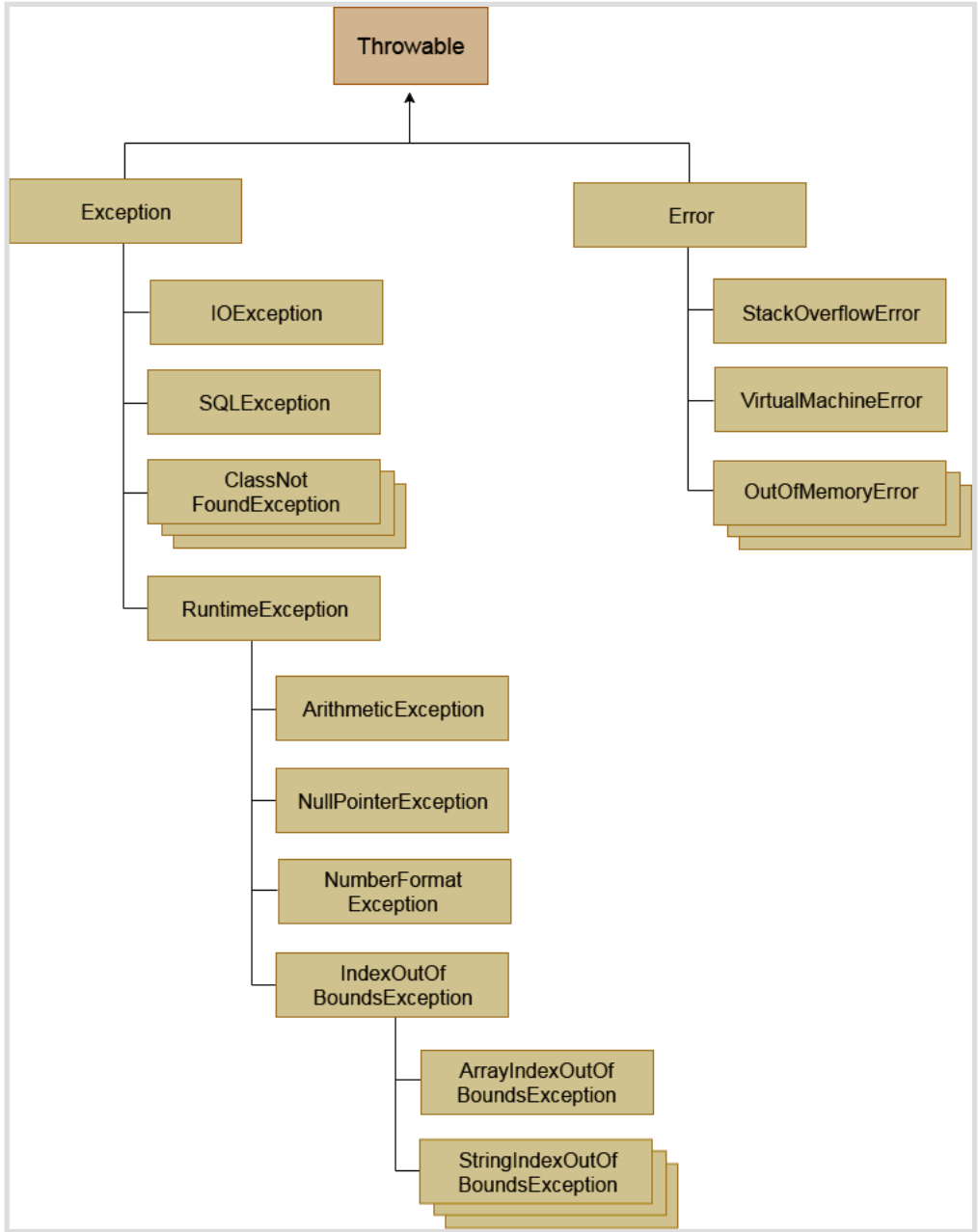
DIFFERENCE BETWEEN ERROR AND EXCEPTION

Error: An Error indicates serious problem that a reasonable application should not try to catch.

Exception: Exception indicates conditions that a reasonable application might try to catch.

IN JAVA…

- When an exceptional condition arises
  - An **object** representing that exception is created
  - Then, the **object** is *thrown* in the method that caused the error
  - The method can choose to handle the exception or pass it on
  - In the end, at some point, exception is *caught* and processed
- Who generates the Exceptions?
  - Can be generated by the Java run-time system
  - Can be manually generated by your code


Throwable
Exception
Error
IOException
SQLException
ClassNot FoundException
RuntimeException
StackOverflowError
VirtualMachineError
OutOfMemoryError
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException

EXCEPTION CLASS

Exception class is used for exceptional conditions that **user programs** should catch

- You will subclass **Exception** to create your own custom exception types
- An important subclass of **Exception** is:
  - **Runtime Exception**: Exceptions of this type are automatically defined for the programs you write
  - Includes: division by zero and invalid array indexing

UNCAUGHT EXCEPTIONS

What happens when you don't handle exceptions in your program ???

```
class uncaught {
public static void main(String args [ ] )
 {
 System.out.println(1/0);
 System.out.println("Hi");
}
```

Exception in thread "main" java.lang.ArithmeticException: /by zero at uncaught.main <uncaught.java:4>

- No compile time error
- Runtime error (Exception)

Example: uncaught.java

UNCAUGHT EXCEPTIONS

When the Java run-time system detects the attempt to divide by zero

- It constructs a new exception object and then *throws* this exception

- What happens next?
  - The execution of **uncaught** stops
  - Exception is *caught* by an exception handler and dealt with immediately

- Where is the exception handler?

UNCAUGHT EXCEPTIONS

Our program should provide the exception handler, catch the exception and process them

- If we haven't supplied any exception handlers of our own
  - The default handler is provided by the Java run-time system
  - Prints a **stack trace** from the point at which the exception occurred
  - **Terminates the program**

EXAMPLE

- The stack trace will always show
  - The sequence of method calls that led up to the error

Example stacktrace:

Exception in thread “main” java.lang.ArithmeticException: / by zero

at div.div1(stacktrace.java:4)at stacktrace.main(stacktrace.java:10)

IN EXAMPLE STACKTRACE

It throws the Exception(ArithmeticException).

Appropriate Exception handler is not found within this method.

JAVA EXCEPTION KEYWORDS

There are five keywords which are used in handling exceptions in Java.

- **try**- The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone
- **catch**- The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
- **finally**-The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.

- **throw**- The "throw" keyword is used to throw an exception
- **throws**- The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

TRY AND CATCH

The default exception handler provided by the Java run-time system is useful for debugging

- Usually, we want to handle an exception ourselves
- **Benefits:**
  - It allows you to fix the error
  - It prevents the program from automatically terminating
- To handle a run-time error
  - Enclose the code that you want to monitor inside a **try** block
  - Immediately following the **try** block, include a **catch** clause
  - Specify the exception type that you wish to catch
  - Example: TCB

TRY AND CATCH CONTD…

In **TCB.java** example:

•**println( )** inside the **try** block is never executed

•Once the **catch** statement has executed

•Program control continues with the next line in the program following the entire **try/catch**

•A **try** and its **catch** statement form a unit

•A **catch** statement cannot catch an exception thrown by another **try** statement

(except in the case of nested **try** statements)

•The statements that are protected by **try** must be surrounded by curly braces

TRY AND CATCH

The goal of most well-constructed **catch** clauses should be

- To resolve the exceptional condition
- Then continue on as if the error had never happened
- Example: ArithmeticException
- Background:

To generate random number: **java.util.Random**

```
Random r = new Random();
int n = r.nextInt();
```