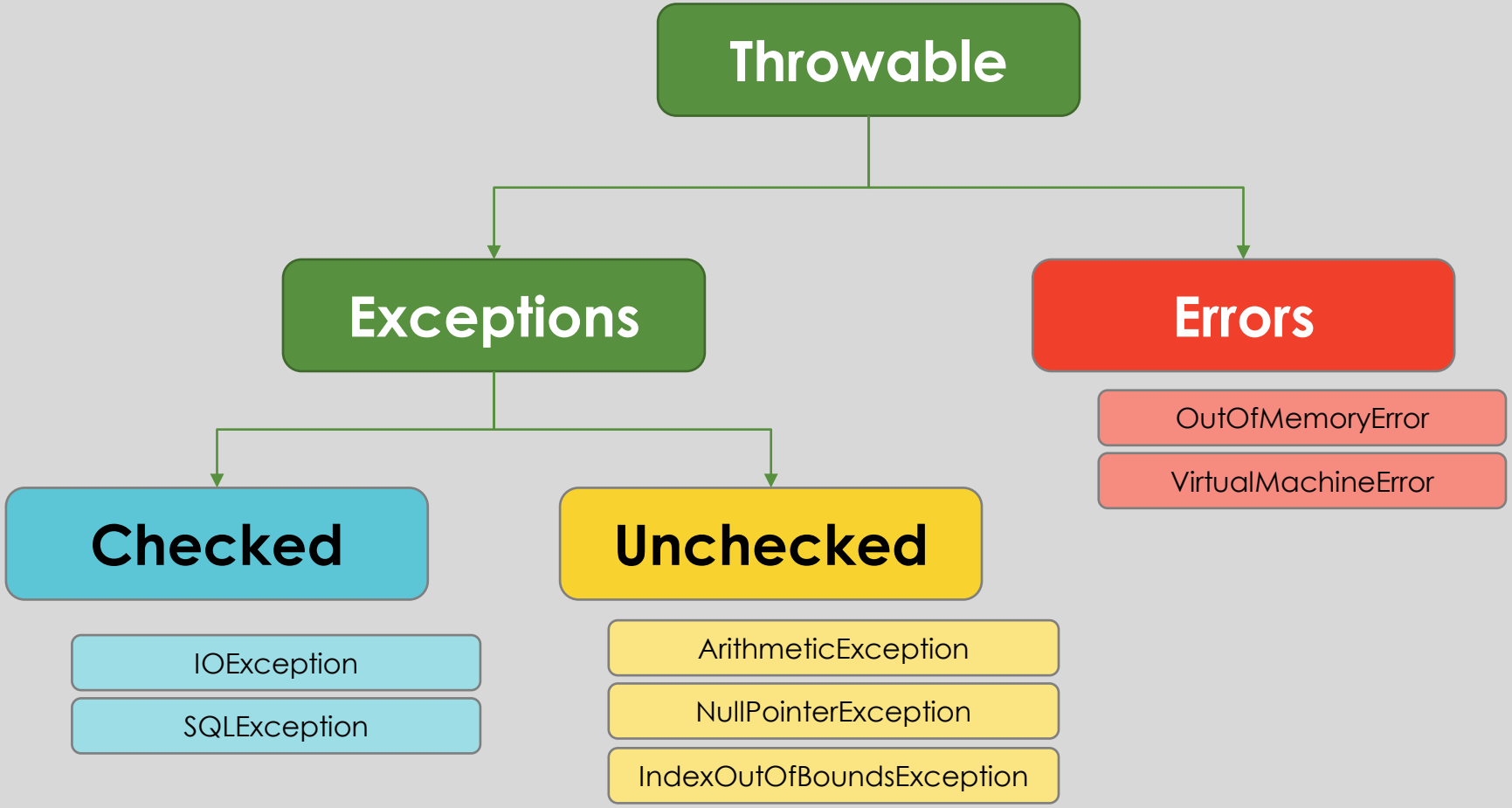

LETS LEARN JAVA
EXCEPTIONS

Training Plan - Intermediate

Topics to be covered
Exception Classification Handling exceptions – try catch finally Multi Try-catch User Defined Exceptions Exception Propagation - Throw and Throws

Exception Classification



Checked Exceptions are checked at compile time, and hence need to be used declaratively

Unchecked exceptions are usually not handled and good coding practices can completely avoid this

Errors are not in your control, but good software design can avoid a few of them

Keywords in exception handling

Keyword	Significance
try	Encloses the piece of code that need to be “tried” to be executed. You expect that there could be an exception thrown in this piece of code depending on situation/data conditions
catch	Defines what type of exception conditions you are handling and what you want to do if such an exception happens
finally	Any cleanup code that you want to execute irrespective of whether there was an exception thrown or not
throw	Used to manually invoke an exception
throws	Used to declare that a method is expected to throw some exception and that it may not be handled. Hence exception is propagated upstream

Exception Example

```java
import java.util.Random;

public class GreetingsService {

    public static void main(String[] args) {
        String greetingMessage = getGreeting();
        System.out.println("Today's greeting of choice " + greetingMessage);

        if (greetingMessage.length() == 11) {
            System.out.println("Its your lucky day too... You get a free coffee to start the day");
        }
    }

    private static String getGreeting() {
        String[] greetings = { "Good Morning", "Hola", "Hi", null, "How are you", "Hey buddy!", null };
        Random random = new Random();
        return greetings[random.nextInt(greetings.length)];
    }
}
```

```
Today's greeting of choice null
Exception in thread "main" java.lang.NullPointerException
        at org.vlearntech.challenges.jlt.GreetingsService.main(GreetingsService.java:11)
```

More Examples of how exceptions are triggered

Let us see how usually we encounter exceptions

Divide a number by Zero to get **ArithmeticException**

```
Integer num = Integer.valueOf("100");
Integer den = Integer.valueOf("0");
System.out.println("Result of Division : " + num / den);
```

Parse a non numeric String into an Integer variable to get **NumberFormatException**

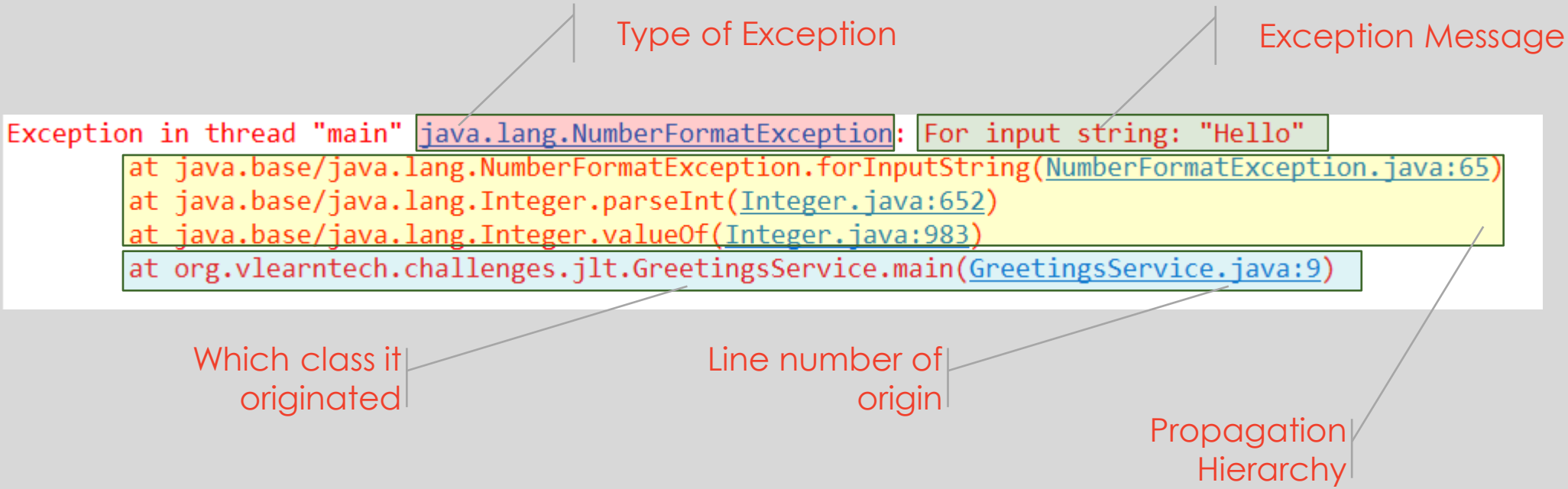
```
Integer num = Integer.valueOf("Hello");
```

Print 3rd element of an array of length 2 to get **ArrayIndexOutOfBoundsException**

```
String[] names = {"Sachin", "Sourav"};
System.out.println(names[2]);
```

Exception Stack Trace

Below message in red that shows up on the screen when there is an exception is called a Stack Trace.



Handling exceptions

Exceptions are handled using the try, catch and finally blocks

```
try {
    Integer num = Integer.valueOf("100");
    Integer den = Integer.valueOf("0");
    System.out.println("Result of Division : " + num / den);
}
```

Try executing this piece of code

```
catch (ArithmeticException e) {
    System.out.println("Arithmetic Exception encountered : " + e.getMessage());
}
```

Handle the exception

```
finally {
    System.out.println("Completed the program");
}
```

Execute this irrespective of whether you got an exception or not

Handling multiple exceptions

Multiple exceptions can also be handled, one after the other.

```
try {
    Integer num = Integer.valueOf("100");
    Integer den = Integer.valueOf("0");
    System.out.println("Result of Division : " + num / den);
} catch (ArithmeticException e) {
    System.out.println("Arithmetic Exception encountered : " + e.getMessage());
} catch (NumberFormatException e) {
    System.out.println("Number Exception encountered : " + e.getMessage());
}
finally {
    System.out.println("Completed the program");
}
```

Specific exceptions should be handled first and then generic exceptions

```
catch (ArithmeticException | NumberFormatException e) {
    System.out.println("Exception encountered : " + e.getMessage());
}
```

A more condensed form of writing same code using the | operator

User-defined exceptions; using throw/throws

You can define your own exception types (remember every class is a type/data type)..

```java
public class InvalidDataException extends Exception {

    public InvalidDataException(String message) {
        super(message);
    }
}
```

```java
public static void main(String[] args) {
    try {
        String numStr = "Hello";
        validateInputData(numStr);
        Integer num = Integer.valueOf(numStr);
    } catch (InvalidDataException e) {
        System.out.println(e.getMessage());
    }
}

static private void validateInputData(String inputNumber) throws InvalidDataException {
    if (inputNumber != null) {
        try {
            Integer.valueOf(inputNumber);
        } catch (NumberFormatException e) {
            throw new InvalidDataException("Invalid input. Expected numeric entry. Received " + inputNumber);
        }
    }
}
```