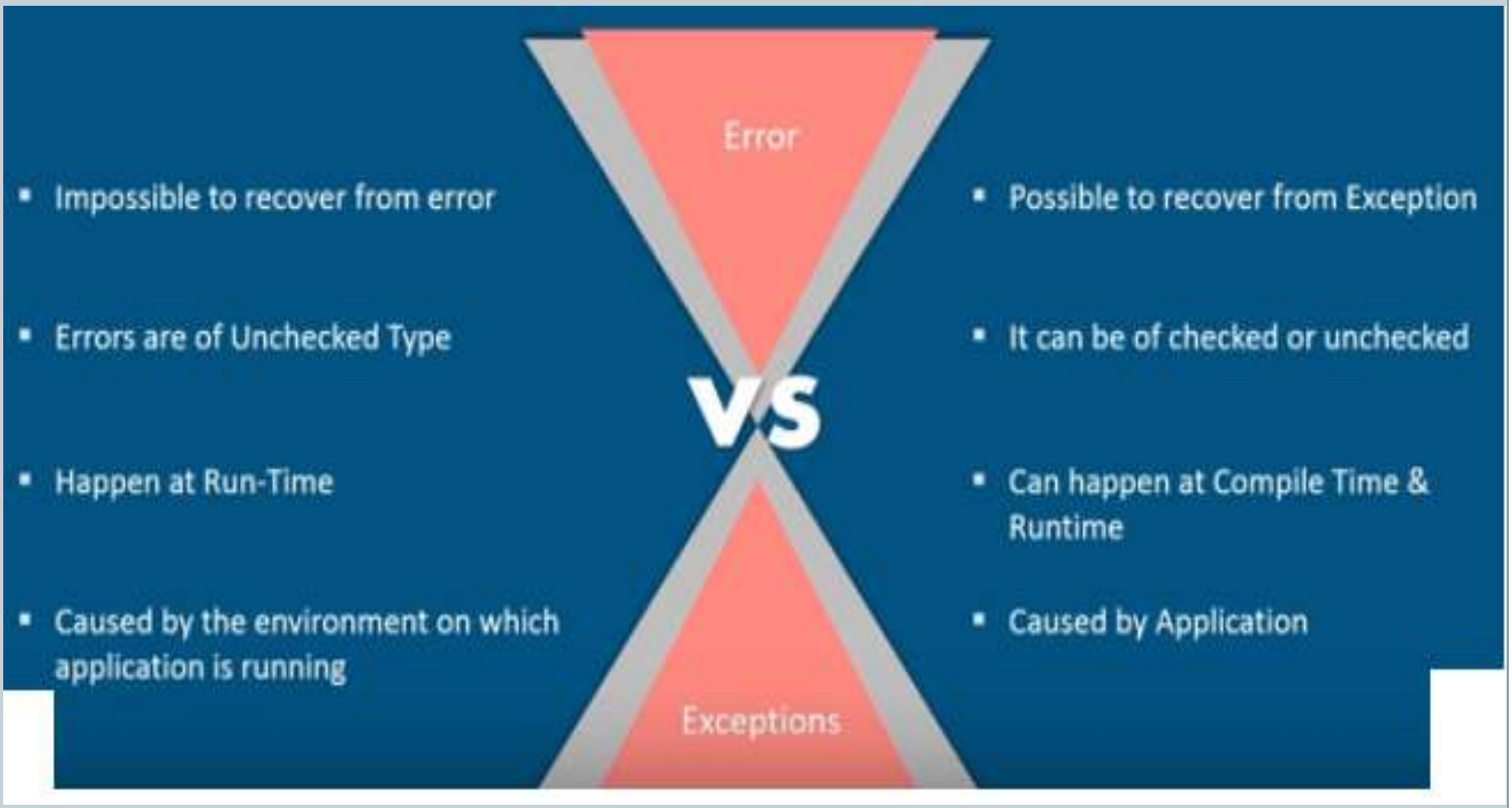
Exception Handling

Unit – 3
Part 1

Exception Handling

Error v/s Exception



What is an Exception?

- An exception is an unexpected event that occurs during runtime and causes normal program flow to be disrupted.

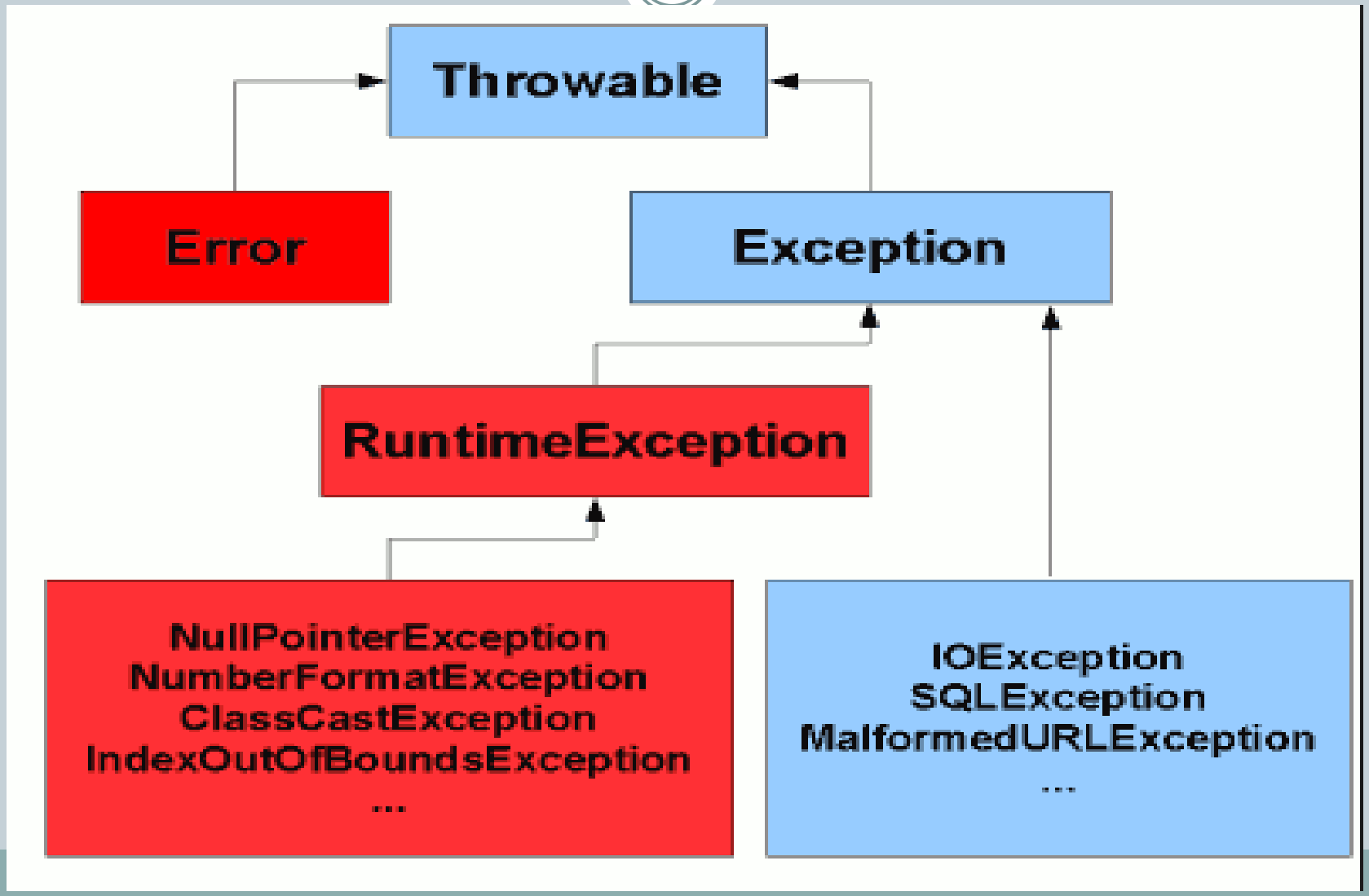
Some common examples:

- Divide by zero errors
- Accessing the elements of an array beyond its range
- Invalid input
- Hard disk crash
- Opening a non-existent file
- Invalid data entry

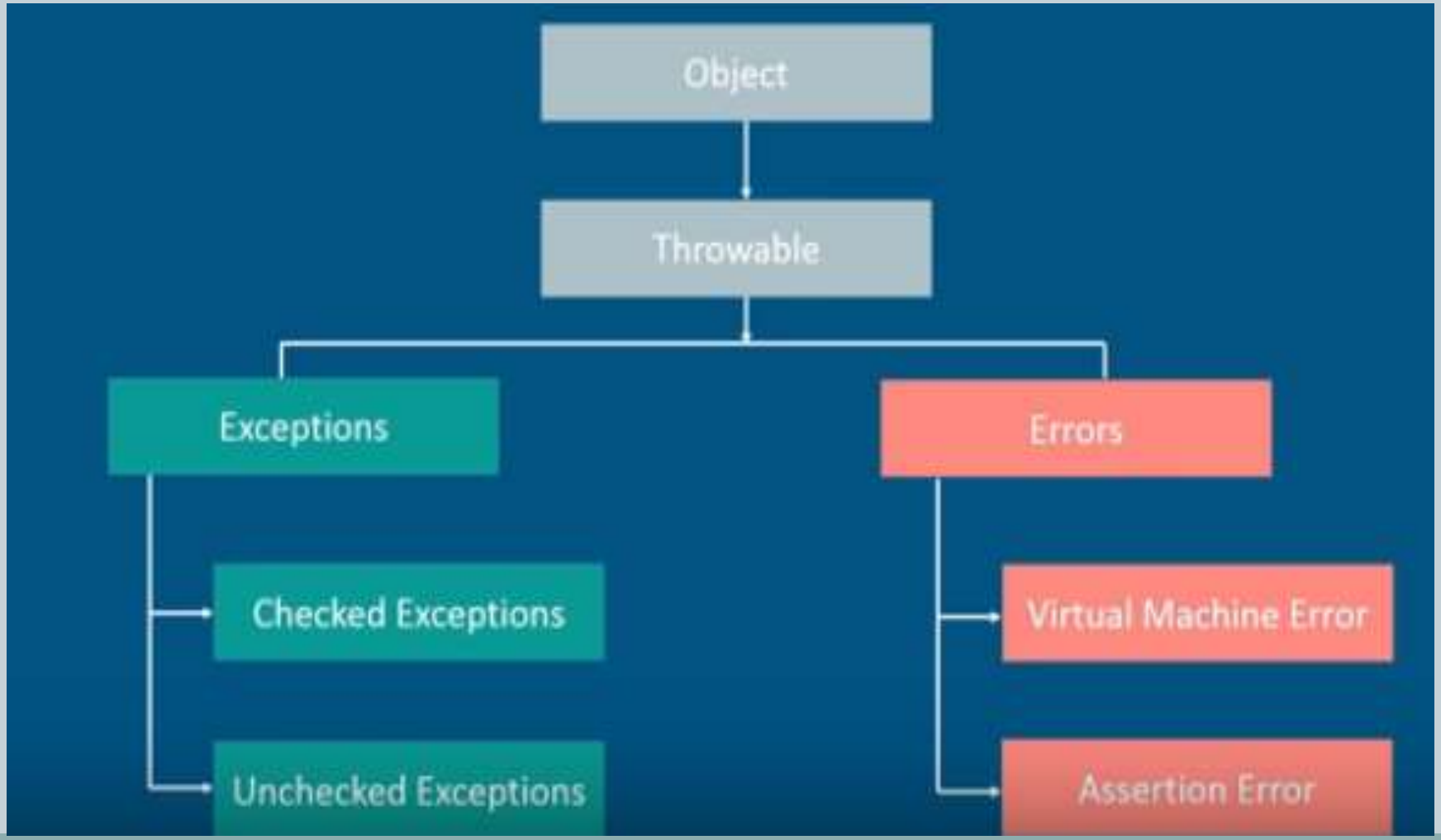
Exceptions Handling

- A powerful mechanism to handle the runtime errors.
- Exception = an abnormal condition.
- Here exception is an event that disrupts the normal flow of the program.
- In Java exception is an **object** which is thrown at runtime.

Java Exception classes



Exception Hierarchy



Exceptions

Checked

- An exception that is checked by the compiler at compilation-time.
- These exceptions cannot simply be ignored, the programmer should handle these exceptions.

Unchecked

- An exception that occurs at the time of execution.
- These are also called as **Runtime Exceptions**.
- Runtime exceptions are ignored at the time of compilation.

Types of Exception

- **Checked Exception:**
  - The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc.
  - Checked exceptions are **checked at compile-time**.
- **Unchecked Exception:**
  - The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException etc.
  - Unchecked exceptions are **checked at runtime**.
- **Error:**
  - Error is irrecoverable e.g. OutOfMemoryError

Checked Exception	Unchecked Exception
Occur at compile time.	Occur at runtime.
Checked by the compiler.	The compiler does not check
Can be handled at the time of compilation.	Cannot be a catch or handle at the time of compilation, because they get generated by the mistakes in the program.
They are the sub-class of the exception class.	Not a part of the Exception class.
Here, the JVM needs the exception to catch and handle.	Here, the JVM does not require the exception to catch and handle.
•Examples of Checked exceptions: 1. File Not Found Exception 2. No Such Field Exception 3. Interrupted Exception 4. No Such Method Exception 5. Class Not Found Exception	•Examples of Unchecked Exceptions: 1. No Such Element Exception 2. Undeclared Throwable Exception 3. Empty Stack Exception 4. Arithmetic Exception 5. Null Pointer Exception 6. Array Index Out of Bounds Exception 7. Security Exception

Keywords of Exception Handling

- There are 5 keywords used in Java exception handling.
  - try
  - catch
  - finally
  - throw
  - throws

try-catch block

- Java try block
  - Java try block is used to enclose the code that might throw an exception.
  - It must be used within the method.
  - Java try block must be followed by either catch or finally block.
- Java catch block
  - Java catch block is used to handle the Exception.
  - It must be used after the try block only.
  - You can use multiple catch block with a single try.

Java try-catch

Syntax of Java try-catch :

```
try
{
        //code that may throw exception
}
catch(Exception_class_Name ref)
{
        -----
}
```

Syntax of try-finally block

```
try
{  //code that may throw exception }
finally
{
        -------
}
```

Without exception handling

```
class Trycatch
{
        public static void main(String args[])
        {
                int data=50/0;//may throw exception
                System.out.println("rest of the code...");
        }
}
```

Output:

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

Using Exception handling

```
class Trycatch_Ex
{  public static void main(String args[])
   {
        try
        {  int data=50/0;   }
        catch(ArithmeticException e)
        {System.out.println(e);}
        System.out.println("rest of the code...");
   }
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Multiple Catch block

- A single try block can have multiple catch blocks. This is required when the try block has statements that generates different types of exceptions.
- If the first catch block contains the **Exception class** object then the subsequent catch blocks are never executed.
- The last catch block in multiple catch blocks must contain the **Exception class** object.

```
try
{
        // Statements which can cause exception
}
catch(Exceptiontype1 e1)
{   ------ }
catch(Exceptiontype2 e2)
{   ------ }
catch(Exceptiontype3 e3)
{   ------ }
```

Example

Nested try block

- Try block within a try block is known as nested try block.
- Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

```
try
 {
        statement 1;
        try {       statement 1;        }
        catch(Exception e)              { }
}
catch(Exception e)
{
}
```

Example

Various Exceptions….

Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
IllegalArgumentException	Illegal argument used to invoke a method.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.

finally block

- finally block is a block that is used to execute important code such as closing connection, stream etc.
- It is always executed whether exception is handled or not.
- It must be followed by try or catch block.

```
try
{
        //Protected code
}
catch(ExceptionType1 e1)
{
        //Catch block
}
catch(ExceptionType1 e1)
{
        //Catch block
}

finally
{
         //The finally block always executes.
}
```

Example

throw v/s throws

Used to throw an exception.

Syntax:

```
void a(){
throw new ArithmeticException("Incorrect");}
```

Used to declare exceptions.

Syntax:

```
void a()throws ArithmeticException {}
```

Throw keyword

- The Java throw keyword is used to explicitly throw an exception.
- We can throw either checked or unchecked exception in Java by throw keyword.
- The throw keyword is mainly used to throw custom exception.
- Syntax :
  - **throw exception;**
- For example :
  - **throw new IOException("sorry device err);**

Example

```
class  TestThrow
{   static void validate(int age)
    {
        if(age<18)
                throw new ArithmeticException("not valid");
        else
                System.out.println("welcome to vote");
  }
     public static void main(String args[])
    {
       validate(13);
       System.out.println("rest of the code...");
     }
}
```

Output: Exception in thread main java.lang.ArithmeticException: not valid

Throws keywords

- The throws keyword is **used to declare an exception**.
- It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
- Syntax :

```
return_type method_name() throws exception class_name
{
        ……..
}
```

Example

Difference between throw & throws

throw keyword	throws keyword
1) throw is used to explicitly throw an exception.	1) throws is used to declare an exception.
2) throw is followed by an instance.	2) throws is followed by class.
3) throw is used within the method.	3) throws is used with the method signature.
4) Checked exception can not be propagated with throw only	4) Checked exception can be propagated with throws
5) You cannot throw multiple exception Ex. throw new IOException("Error...")	5) You can declare multiple exception e.g. public void method()throws IOException,SQLException.

Java Custom Exception

- If you are creating your own Exception that is known as custom exception or user-defined exception.
- Java custom exceptions are used to customize the exception according to user need.
- By the help of custom exception, you can have your own exception and message.
- Example

```
/* User defined excetpion / Custom exception */
class InvalidAgeExp extends Exception
{
    InvalidAgeExp(String s)
    {
        super(s);
    }
}
class CustomException
{
    static void validate(int age)throws InvalidAgeExp
  {
                if(age<18)
                        throw new InvalidAgeExp("not valid");
              else
                        System.out.println("welcome to vote");
    }
    public static void main(String args[])
     {
        try
         {   validate(13);              }
          catch(Exception m) {System.out.println("Exception occured: "+m);        }
    System.out.println("rest of the code...");
    }
}
```

final v/s finally v/s finalize