Java Exception Handling

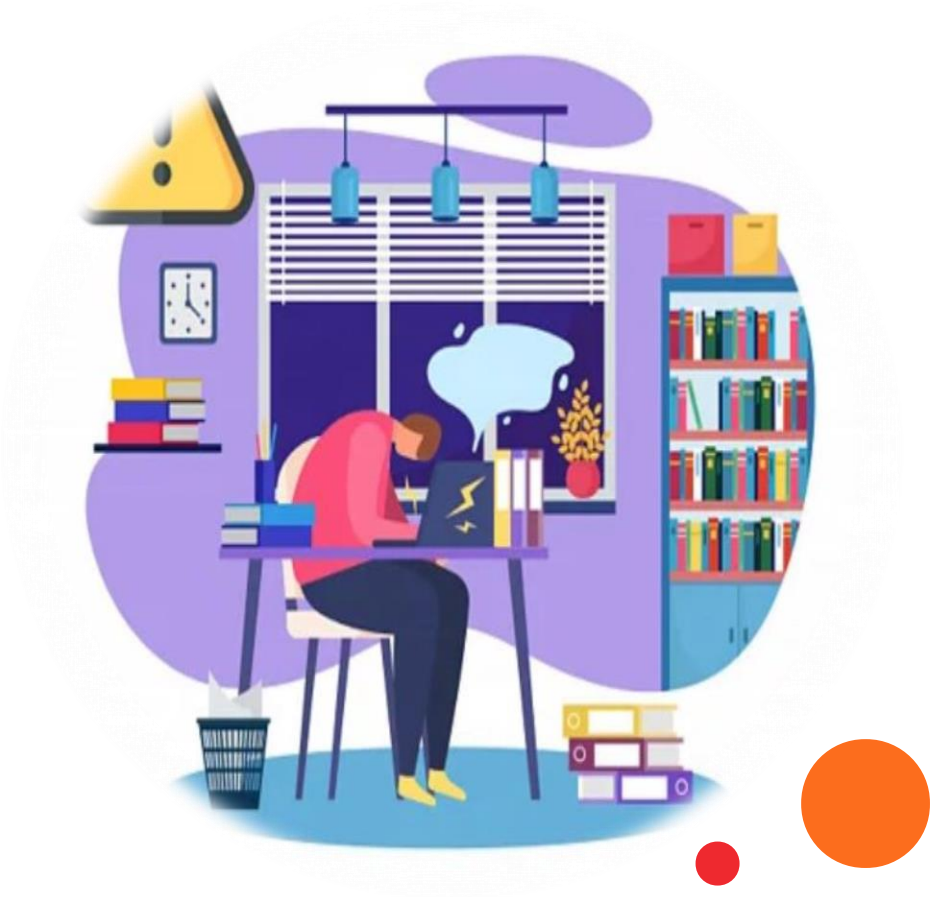
Hexavarsity

Objective

- Exception Introduction
- Exception types
- Exception Handling Mechanism
- Custom Exception

Exception Introduction

What is Exception?

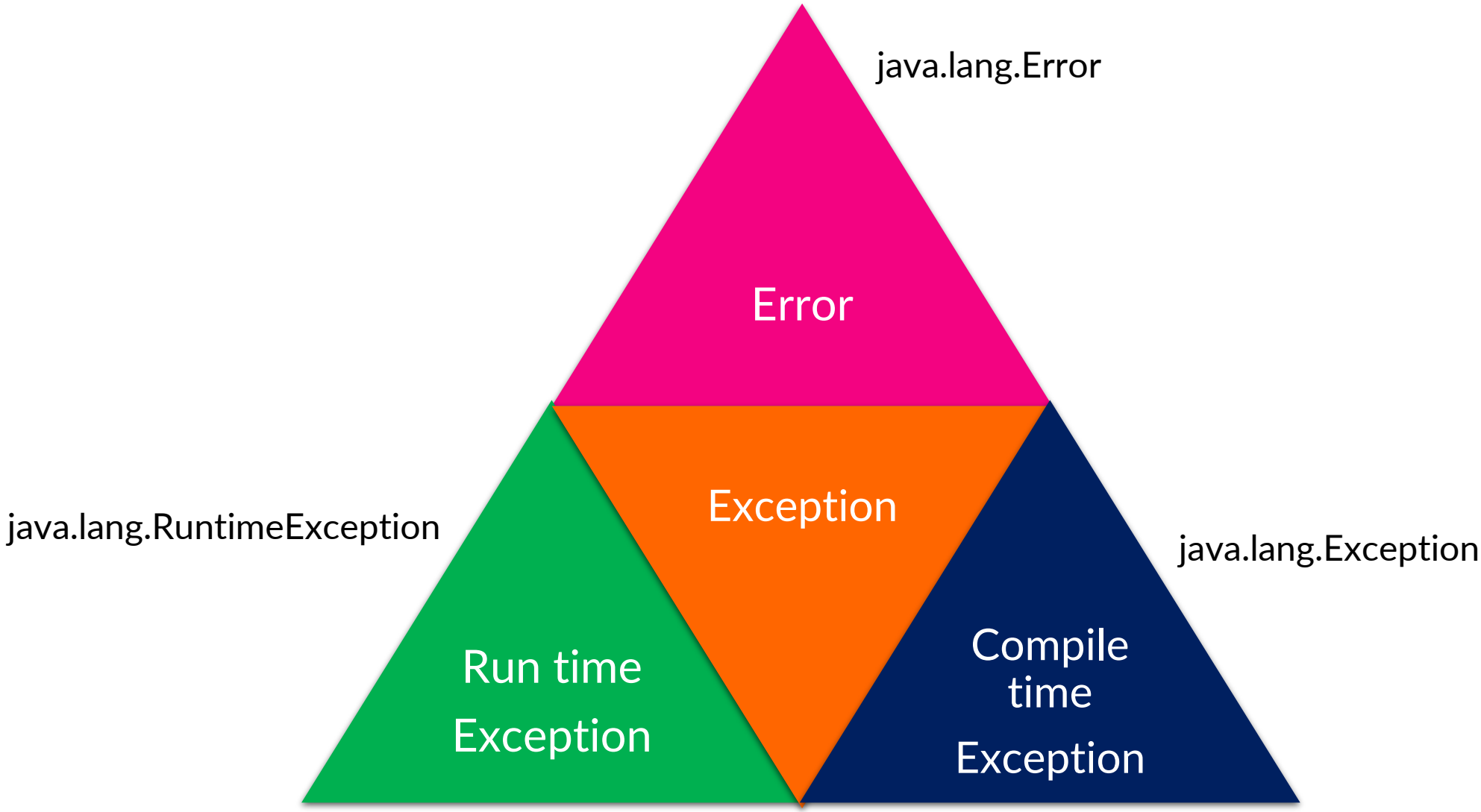


- An Exception is an unwanted operation in a program that interrupts the normal flow of program execution.
- When an exception occurs program execution gets terminated and print Exception message.
- Following reason for Exception.
  1. A user has entered an invalid data
  2. File not found
  3. The JVM has run out of a memory
  4. A network connection has been lost in the middle of communications

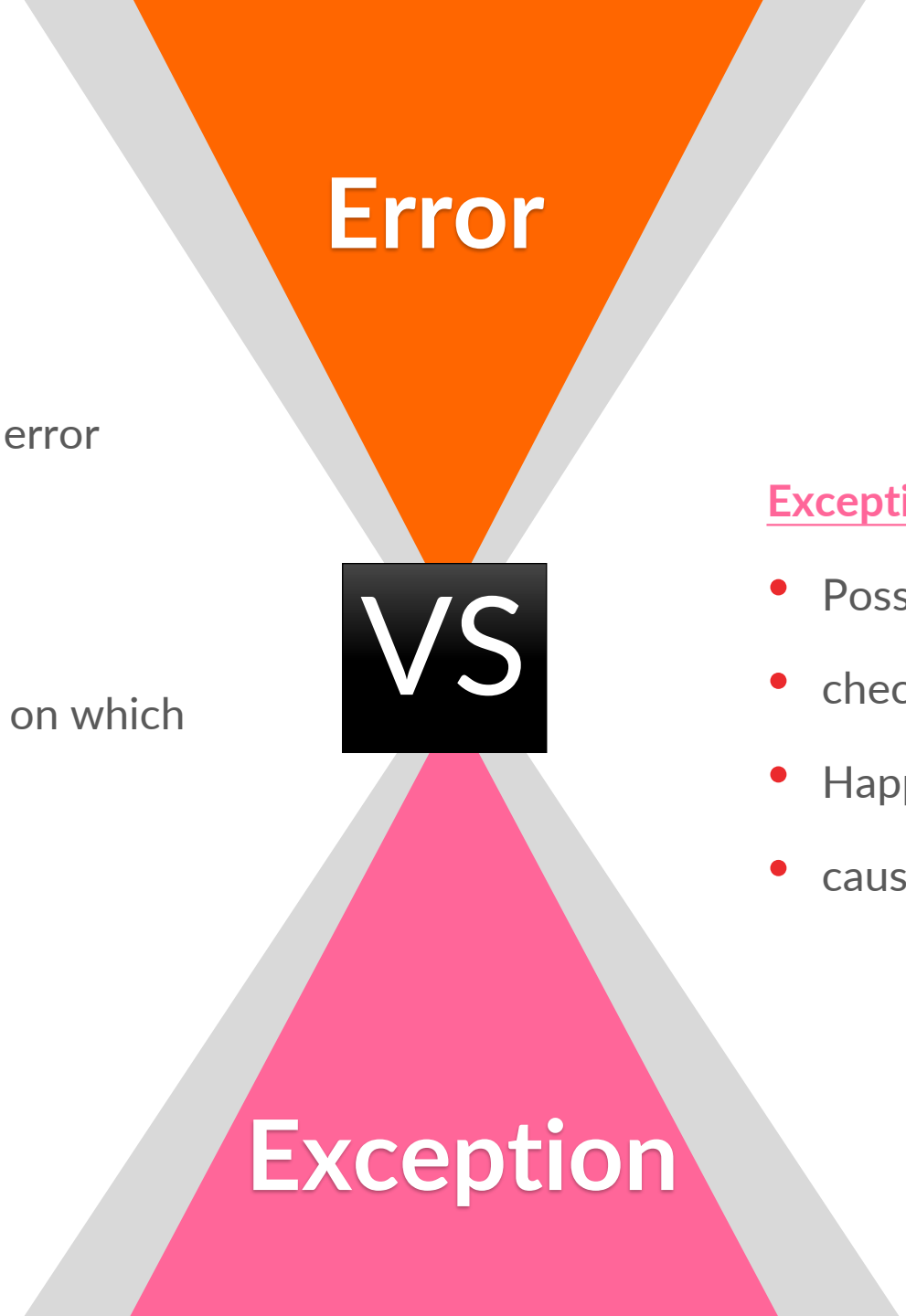
Realtime Example for exception

Realtime Example for Exception Handling

Types of exceptions



Error vs Exception



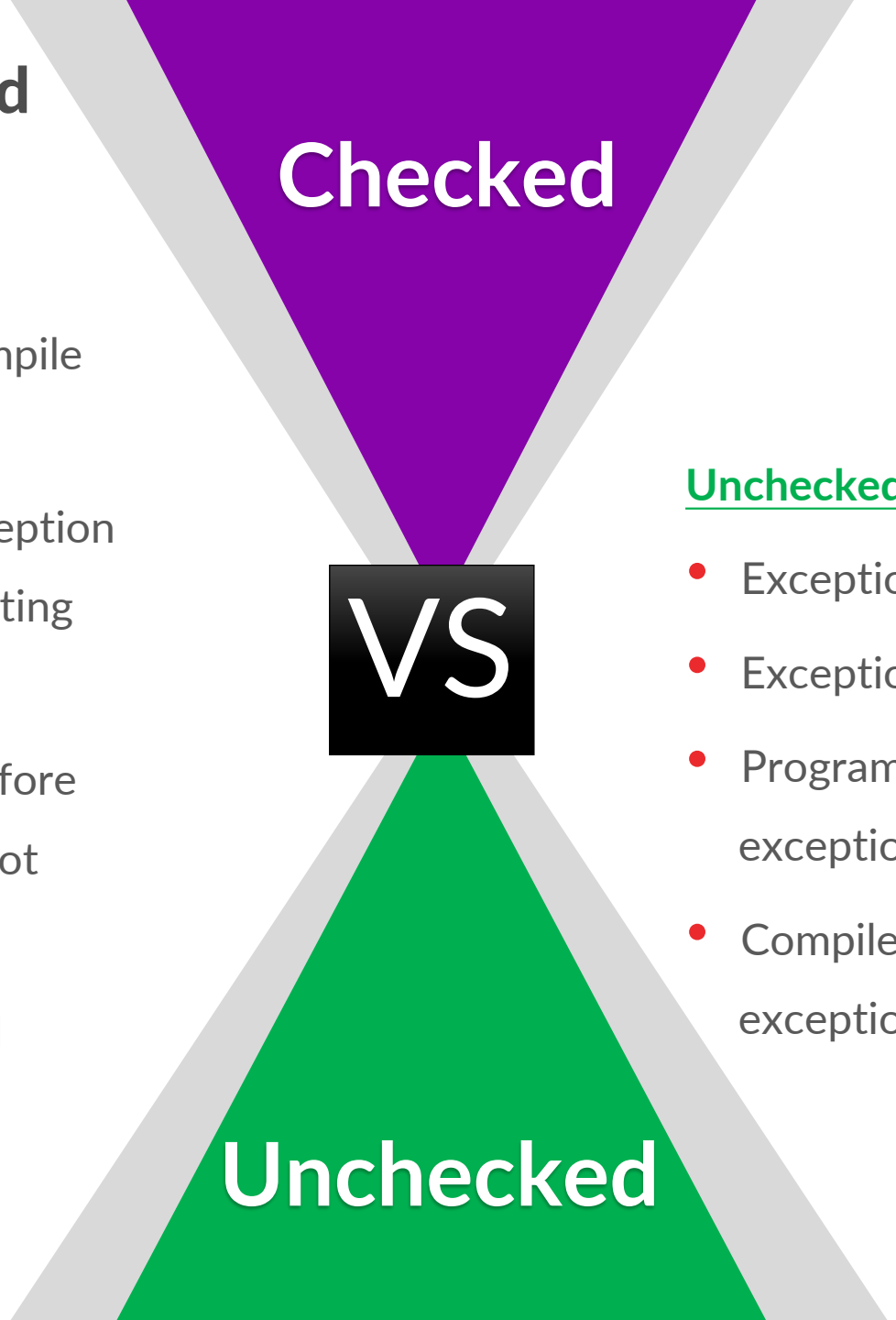
Error

- Impossible to recover from error
- Error are unchecked
- Happen at Run-time
- caused by the environment on which application running

Exception

- Possible to recover from exception
- checked or unchecked
- Happen @ Run,compile time
- caused by the application

Checked vs Unchecked



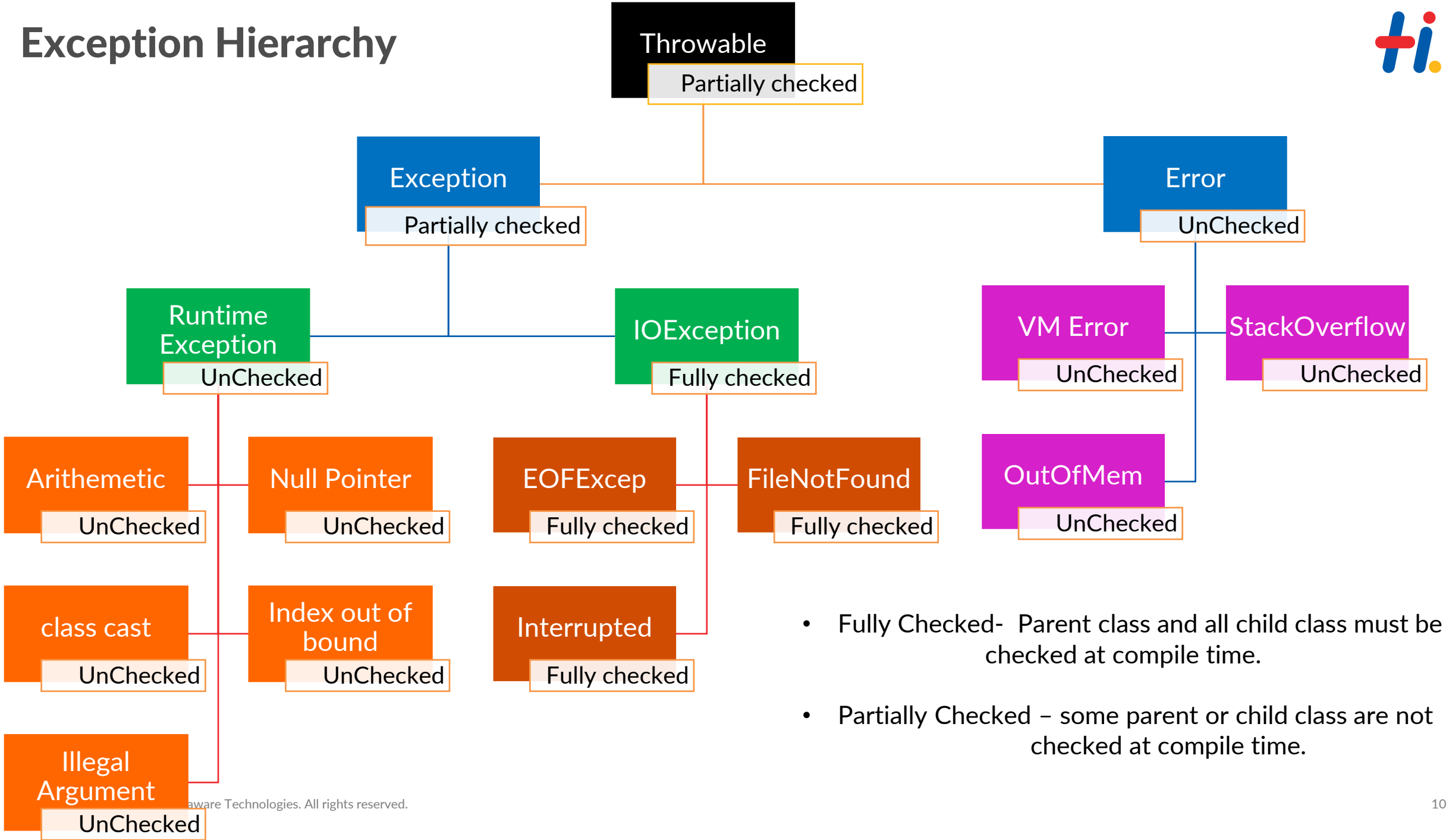
Checked

- Exception that occurs at compile time [checked by compiler]
- Compiler have report of exception that will happen while executing program.
- Programmer must handle before compiling the program, cannot ignored by programmer
- Eg: IOException, interrupted exception, file not found

Unchecked

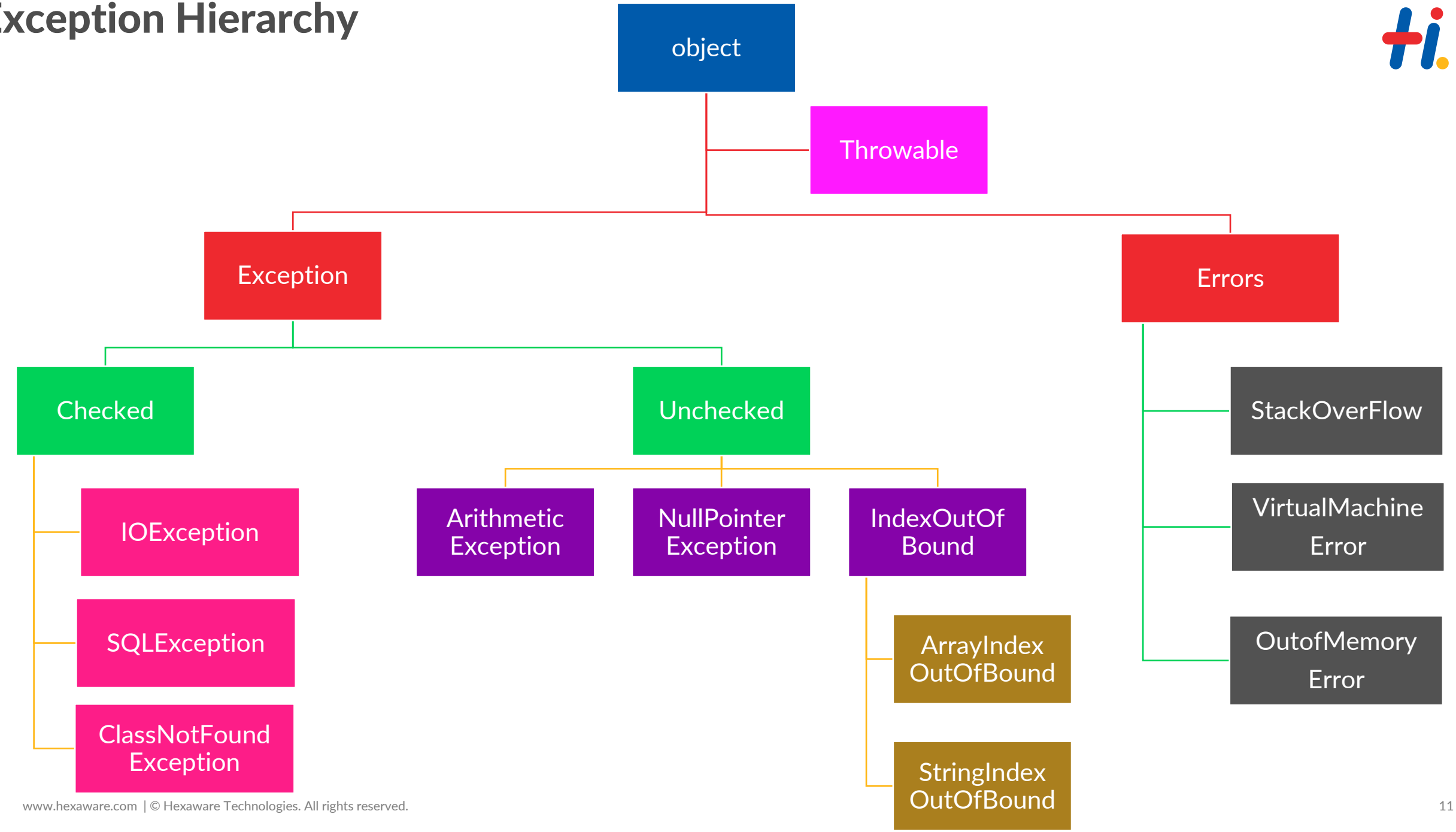
- Exception that occurs at run time.
- Exception not checked by the compiler.
- Programmer can ignore to handle this exception.
- Compiler does not have report of this exception.

Exception Hierarchy



- Fully Checked- Parent class and all child class must be checked at compile time.
- Partially Checked – some parent or child class are not checked at compile time.

Exception Hierarchy



Unchecked Exceptions

Null Pointer Exception Example

```
class NullPtrException
{
        public static void main(String[] args)
        {
                String a = null;
                a.length();
        }
}
```

Example:

```
import java.io.*;
class UserExcep
{
        static ArithmeticException e;
        public static void main(String[] args)
        {
                throw e;
        }
}
```

Example:

```
class Student
{       int a=5;                }
class NullPtrException
{
        public static void main(String[] args)
        {
                Student std = null;
                std.a=67;
        }
}
```

Example:

```
Exception in thread "main" java.lang.NullPointerException
        at NullPtrException.main(NullPtrException.java:6)
```

NumberFormatException Example

```
class NoFormatExcep
{
        public static void main(String[] args)
        {
                int a = Integer.parseInt("56");
                System.out.println(a);
                int b = Integer.parseInt(“Hexaware");

        }
}
```

Example:

```
56
Exception in thread "main" java.lang.NumberFormatException: For input string: “Hexaware“     at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at NoFormatExcep.main(NoFormatExcep.java:7)
```

ArithmeticException Example

```
class ArithmeticExcep
{
        public static void main(String[] args)
        {
                int a=50,b=0,c;
                c = a / b;
        }
}
```

Example:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ArithemeticExcep.main(ArithemeticExcep.java:6)
```

IndexOutofBounds Exception Example

Example:

```
class ArrayIndxOutofBound
{
        public static void main(String[] args)
        {
                int[] arr = new int[5];
                arr[5] = 30;
        }
}
```

```
Array index out of bound example
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
        at
ArrayIndxOutofBound.main(ArrayIndxOutofBound.java
:7)
```

Example:

```
class StringIndexOtBoundExcep
{
        public static void main(String[] args)
        {
                String s = "welcome";
                s.charAt(7);
        }
}
```

```
Array index out of bound example
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
        at
ArrayIndxOutofBound.main(ArrayIndxOutofBound.java
:7)
```

ClassCastException Example

Example:

```
class Car{}

class Benz extends Car{}

class Maclaren extends Benz{}

class ClassCastExcep
{
        public static void main(String[] args)
        {
        Maclaren sporty = (Maclaren) new Benz();

        }
}
```

```
Exception in thread "main" java.lang.ClassCastException: Benz cannot
be cast to Maclaren
        at ClassCastExcep.main(ClassCastExcep.java:9)
```

IllegalArgumentException Example

Example:

```
class IllegalArgException
{
        public static void main(String[] args)
        {
                Thread t = new Thread();
                t.setPriority(11);
        }
}
```

```
Exception in thread "main" java.lang.IllegalArgumentException
        at java.lang.Thread.setPriority(Unknown Source)
        at IllegalArgException.main(IllegalArgException.java:6)
```

Demo

Checked Exceptions

IOException Example

Example:

```
import java.io.*;
class IoExcep {
        public static void main(String[] args) {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                String s = br.readLine();
        }
}
```

```
IoExcep.java:7: error: unreported exception IOException; must be caught or declared to be thrown
        String s = br.readLine();
```

InterruptedException Example

Example:

```
class InterruptedExceptionDemo
{
        public static void main(String[] args)
        {
                Thread.sleep(1000);
        }
}
```

```
IntrpdExcep.java:5: error: unreported exception InterruptedException; must be caught or declared to be thrown
                Thread.sleep(1000);
```


FileNotFoundException Example

Example:

```
import java.util.*;
import java.io.*;
class FileNotFound
{
        public static void main(String[] args)
        {
        Scanner sc = new Scanner(new File("file1.txt"));

        }
}
```

```
FileNotFound.java:7: error: unreported exception
FileNotFoundException; must be caught or
                                        declared to be
thrown
        Scanner sc = new Scanner(new File("file1.txt"));
```

ClassNotFoundException Example

Example:

```
class ClassNotFound
{
        public static void main(String[] args)
        {

        Class.forName("oracle.jdbc.driver.OracleDriver");

        }
}
```

```
FileNotFound.java:7: error: unreported exception
FileNotFoundException; must be caught or
                                        declared to be
thrown
        Scanner sc = new Scanner(new File("file1.txt"));
```


NoClassDefFoundError Example

Example:

```
class Employee { }
class ClassNotFound
{
        public static void main(String[] args)
        {
                //Class.forName("oracle.jdbc.driver.OracleDriver");
                Employee emp = new Employee();
        }
}
```

- When you compile the above program, two .class files will be generated.
- One is Employee.class and another one is ClassNotFound.class.
- If you delete the Employee.class file and run the ClassNotFound.class file, Java Runtime System will throw NoClassDefFoundError like below:

```
Exception in thread "main" java.lang.NoClassDefFoundError: Employee
    at ClassNotFound.main(ClassNotFound.java:10) Caused by: java.lang.ClassNotFoundException: Employee
    at java.net.URLClassLoader.findClass(Unknown Source)
    at java.lang.ClassLoader.loadClass(Unknown Source)
    at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
    at java.lang.ClassLoader.loadClass(Unknown Source)
```


Exception Hierarchy

ClassNotFoundException	NoClassDefFoundError
It is an exception. It is of type **java.lang.Exception.**	It is an error. It is of type **java.lang.Error.**
It occurs when an application tries to load a class at **run time** which is not updated in the classpath.	It occurs when java runtime system doesn't find a class definition, which is **present at compile time, but missing at run time.**
It is thrown by the application itself. It is thrown by the methods like **Class.forName(), loadClass() and findSystemClass().**	It is thrown by the Java Runtime System.

Demo on Checked Exceptions

Runtime Error

OutOfMemoryError Example

Example:

```
import java.util.*;
class StackOvrFlow
{
        public static void main(String[] args)
        {
                ArrayList al = new ArrayList();
                while(true) al.add("ARR");
        }
}
```

```
Exception in thread "main" java.lang.OutOfMemoryError:
Java heap space
        at java.util.Arrays.copyOf(Unknown Source)
        at java.util.Arrays.copyOf(Unknown Source)
        at java.util.ArrayList.grow(Unknown Source)
        at java.util.ArrayList.ensureExplicitCapacity(Unknown
Source)
        at java.util.ArrayList.ensureCapacityInternal(Unknown
Source)
        at java.util.ArrayList.add(Unknown Source)
        at StackOvrFlow.main(StackOvrFlow.java:8)
```

StackOverflowError Example

Example:

```
class ClassNotFound
{
        public static void main(String[] args)
        {

        Class.forName("oracle.jdbc.driver.OracleDriver");

        }
}
```

```
Exception in thread "main" java.lang.StackOverflowError
at StackOvrFlow.motherBoard(StackOvrFlow.java:9)
        at StackOvrFlow.laptop(StackOvrFlow.java:5)
```

Exception Handling

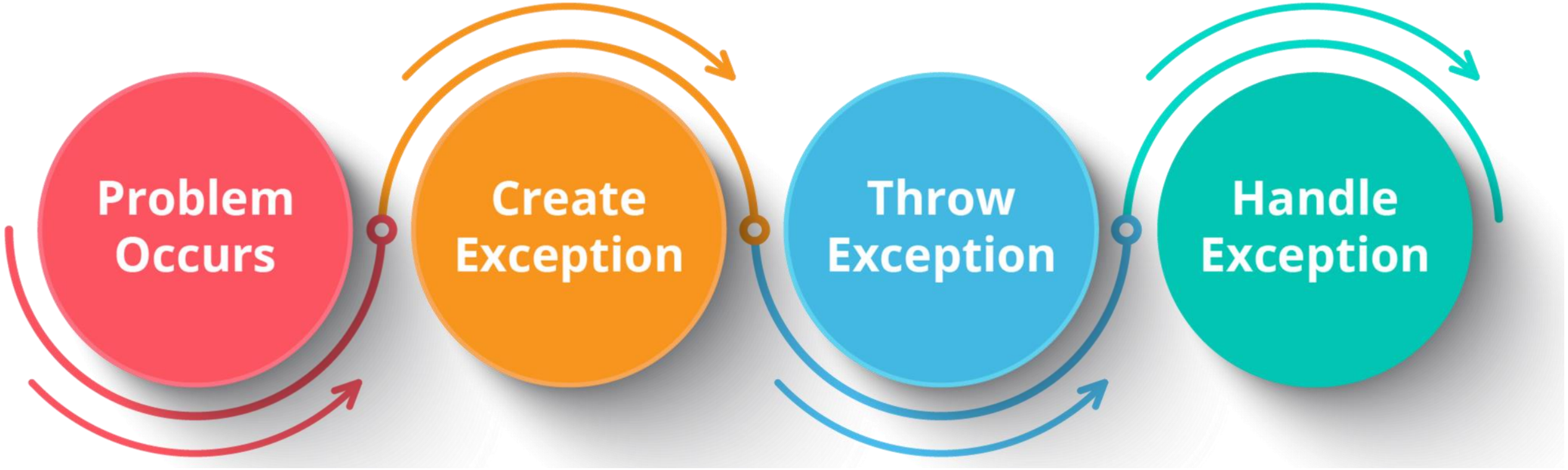
Exception Handling Mechanism

- Exception Handling Provide alternate way to exit the program execution when any exception occurs is called Exception Handling.

- **Default Exception Handling:**
  - if an exception has occurred inside a method, the method creates and handover an Exception Object to JVM.
  - **Exception Object contains:**
    1. Name of the exception
    2. Description of the exception,
    3. Current state of the program where exception has occurred

- Creating the Exception Object and handling it to JVM is called throwing an Exception.

Exception Handling Mechanism



- Java provides various option to handle the Exceptions like:
  - Try
  - catch
  - finally
  - throw
  - throws

Exception Handling Mechanism

- TRY Block
- CATCH
- FINALLY
- THROW
- THROWS

- Try block contains a set of statements where an exception can occur
- A try block must be followed by catch blocks or finally block or both.

Syntax:

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

Exception Handling Mechanism

TRY Block

CATCH

FINALLY

THROW

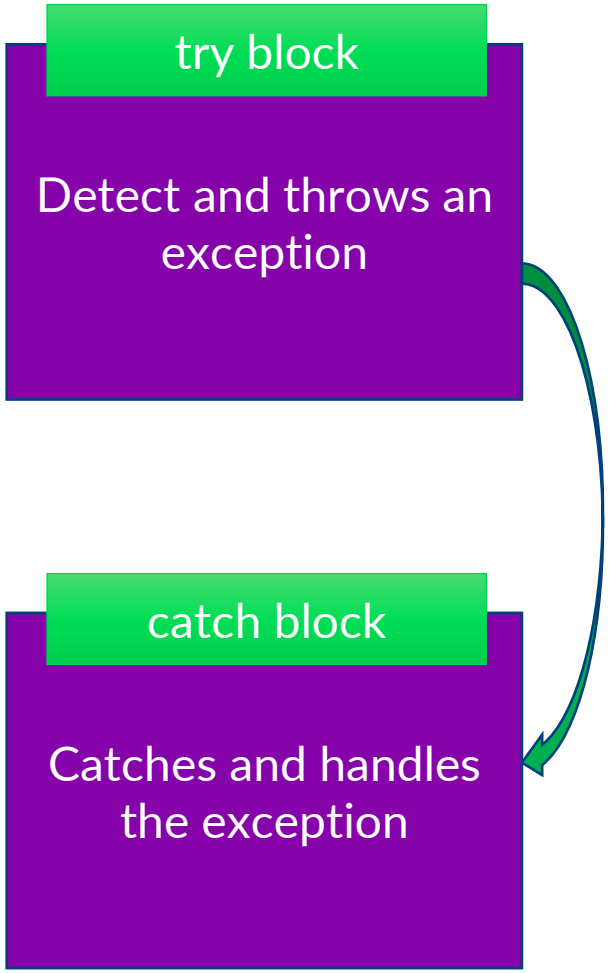
THROWS

- A catch block is responsible to handle the exceptions
- Catch block must follow the try block
- A single try block can have several catch blocks
- When an exception occurs in a try block, the corresponding catch block that handles that particular exception executes

Single TRY, CATCH

Example:

```
class SingleTryCatch
{
        public static void main(String[] args)
        {
                try
                {
                        int a = 45 / 0;
                        System.out.println(a);
                }
                catch(ArithmeticException e)
                {
                        System.out.println("No divide by 0");
                }
        }
}
```



Output:
No divide by 0

Nested Try, CATCH

Example:

```
class NestedTryCatch {
    public static void main(String[] args) {
        int[] arr = new int[5];
        try {
            try {
                int a = 45 / 0;
                System.out.println(a);
            } catch (ArithmeticException e) {
                System.out.println("No divide by 0");
            }
            try {
                arr[5] = 45;
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("Try to access invalid index, array size is " + arr.length);
            }
        } catch (Exception e) {
            System.out.println("Outer try exceuted");
        }
    }
}
```

No divide by 0
Try to access invalid index, array size is 5

Multiple Catch

Example:

```
class MultipleCatch {
   public static void main(String[] args)
   {
         try {   int a = 45 / 0; }

         catch(ArrayIndexOutOfBoundsException e)
         {
            System.out.println("Try to access invalid index");
         }
         catch(ArithmeticException e)
         {
            System.out.println("No divide by 0");
         }
         catch(Exception e)  {
             System.out.println(“Exception");
         }
   }
}
```

Output:
No divide by 0

MultipleCatch (Predict the output)

Example:

```
class MultipleCatch
{
   public static void main(String[] args)
   {
         try
         {
            int a = 45 / 0;
         }
         catch(ArithmeticException e)
         {
            System.out.println("No divide by 0");
         }
         catch(ArithmeticException e)
         {
            System.out.println("No divide by 0");
         }
   }
}
```

```
error: exception ArithmeticException has already been caught
                  catch(ArithmeticException e)
                  ^
1 error
multiple catch should not catch same exception
```


Multiple Catch(Predict the output)

Example:

```
class MultipleCatch {
    public static void main(String[] args) {
        try {
            int a = 45 / 0;
        }
        catch (Exception e)
        {
            System.out.println("Outer try exceuted");
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Try to access invalid index");
        }
        catch (ArithmeticException e)
        {
            System.out.println("No divide by 0");
        }
    }
}
```

Compile time error

```
error: exception ArrayIndexOutOfBoundsException
has already been caught

catch(ArrayIndexOutOfBoundsException e)
                    ^
error: exception ArithmeticException has already been
caught
                        catch(ArithmeticException e)
                                        ^
2 error
```

multiple catch block without maintaining the order of exception

Demo

Exception Handling Mechanism

TRY Block

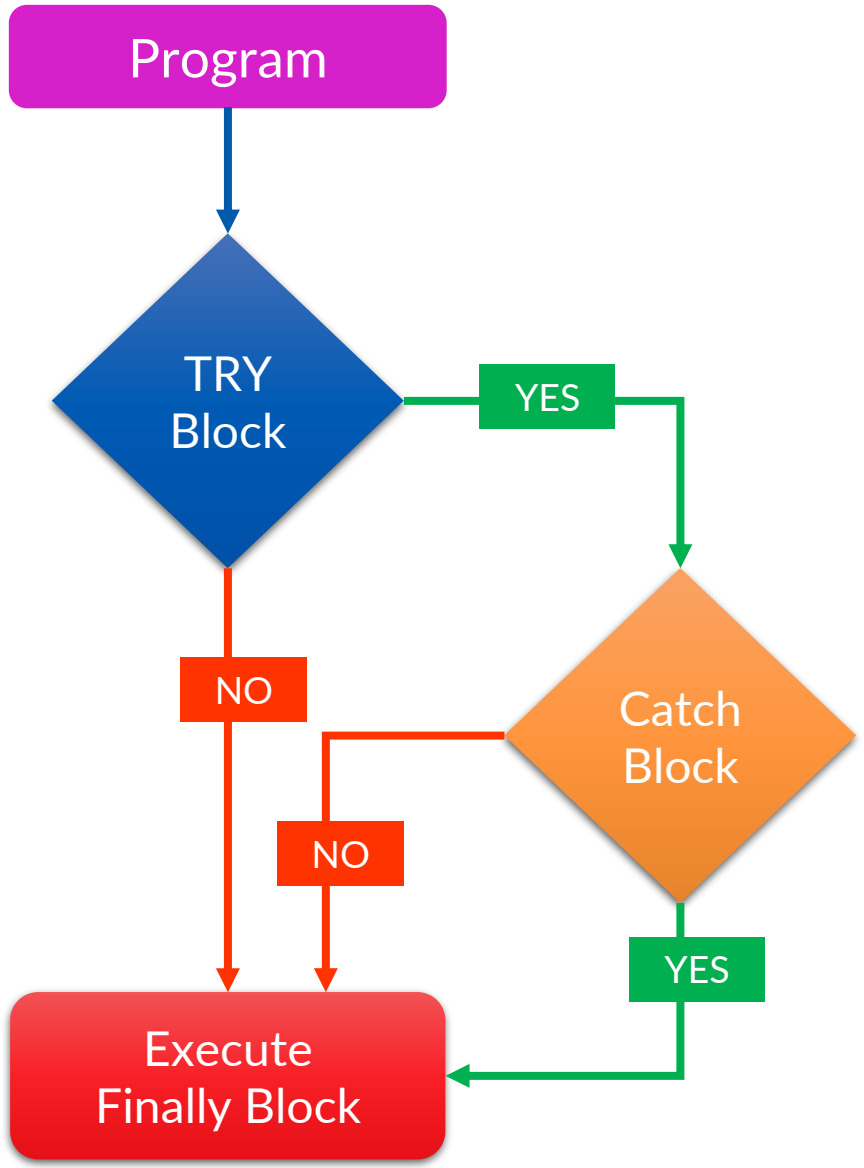
CATCH

FINALLY

THROW

THROWS

- Finally, block is used to execute important code such as closing DB connection, IO stream etc.
- Finally block statement will always execute, either an exception occurs in the try block or not
- Finally block follows try or catch block.



Case 1:Finally follow Try

Example:

```
class TryFinally
{
   public static void main(String[] args)
   {
      try {   int a = 50/0;   }
      finally
      {
         System.out.println("Try without catch\nTry with
finally");
      }
   }
}
```

```
Try without catch
Try with finally
Exception in thread "main" java.lang.ArithmeticException: /
by zero
      at TryFinally.main(TryFinally.java:7)
```

Case 2: exception occurs and not handled

Example:

```
class FinallayWOExcep{
   public static void main(String args[])
   {
      try      {
         int data=25/0;  //cause Arithemetic Exception
      }
      // No catch block handle Arithmetic Exception
      catch(NullPointerException e)
      {
         System.out.println(e);
      }
      finally
      {
         System.out.println("finally is always executed");
      }
}
```

```
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: /
by zero
atFinallayWOExcep.main(FinallayWOExcep.java:7)
```

Case 3: Exception occurs and handled

Example:

```
class FinallayWOExcep
{
   public static void main(String args[])
   {
      try
      {
         int data=25/0;
         System.out.println(data);
      }
      catch(ArithmeticException e)
      {
         System.out.println(e);
      }
      finally
      {
         System.out.println("finally block is always
executed");
      }
}
```

```
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: /
by zero
atFinallayWOExcep.main(FinallayWOExcep.java:7)
```


Exception Handling Mechanism

TRY Block

CATCH

FINALLY

THROW

THROWS

- We can define our own set of conditions or rules and throw an exception explicitly using throw keyword.
- For example, we can throw ArithmeticException when we divide number by 5 or other number.

Syntax

```
throw new exception_class("error message");
```

Throw keyword example – ArithmeticException / by 5

Example:

```
class Division
{
   static void divide(int a,int b)
   {
      if(b==5)
         throw new ArithmeticException("No / by 5");
      else
         System.out.println(a/b);
   }
}
class ThrowKeyword
{
   public static void main(String[] args)
   {
      Division.divide(10,5);
   }
}
```

```
Exception in thread "main" java.lang.ArithmeticException:
No / by 5
        at Division.divide(ThrowKeyword.java:6)
        at ThrowKeyword.main(ThrowKeyword.java:15)
```

Note

```
if(b==5)
{
throw new ArithmeticException("No / by 5");
System.out.print("welcome");
}
```

compiler error: unreachable statement
System.out.println("Welcome");
>>>throw must be last statement <<<
>>>after throw statement print line is there, so error<<<

Exception Handling Mechanism

TRY Block

CATCH

FINALLY

THROW

THROWS

- Throws keyword is used for handling checked exceptions .
- By using throws we can declare multiple exceptions in one go.
- The caller to these methods has to handle the exception using a try-catch block.

Syntax

```
return_type method_name(parameters)
throws exception1, exception2(opt),...
```


Throw keyword example – ArithmeticException / by 5

Example:

```
import java.util.*;
import java.io.*;
class FileNotFound
{
   public static void main(String[] args)
   {
   Scanner sc = new Scanner(new File("file1.txt"));
   }
}
```

```
FileNotFound.java:7: error: unreported exception
FileNotFoundException; must be caught or
                                                declared to be
thrown
            Scanner sc = new Scanner(new File("file1.txt"));
```

Example:

```
import java.util.*;
import java.io.*;
class FileNotFound
{
   public static void main(String[] args) throws
                                        FileNotFoundException
   {
   Scanner sc = new Scanner(new File("file1.txt"));
   }
}
```

compile time exception is not thrown

Demo

throws keyword: points to remember

- throws keyword is required only for checked exception
- usage of throws keyword for unchecked exception is meaningless.
- throws keyword is required only to convince compiler
- usage of throws keyword does not prevent abnormal termination of program.
- By the help of throws keyword we can provide information to the caller of the method about the exception.

throw vs throws

- Used to explicitly throw an exception
- Followed by an instance
- Used within a method
- Cannot throw multiple exceptions

- Used to declare an exception in method
- Followed by a class
- Used with a method signature
- Can throw multiple exceptions

final vs finally vs finalize

final	finally	finalize
It is a keyword.	It is a block.	It is a method.
Used to apply restrictions on class, methods & variables.	Used to place an important code.	Used to perform clean-up processing just before the object is garbage collected.
final class can’t be inherited, method can’t be overridden & the variable value can’t be changed.	It will be executed whether the exception is handled or not.	–

User defined exception

User Defined Runtime Exception using Throw Keyword

```
class VoterEligility extends RuntimeException
{
   VoterEligility(String s)
   {
      super(s);
   }
}
class UserDefinedExcep
{
   public static void main(String[] args)
   {
      int age = Integer.parseInt(args[0]);
      if(age<18)
         throw new VoterEligility("Not Eligible for vote");
      else
         System.out.println(age+" Eligible for vote");
   }
}
```

Example:

```
Exception in thread "main" VoterEligility: Not Eligible for vote
        at UserDefinedExcep.main(UserDefinedExcep.java:14)
```

User Defined Compile time Exception using Throw Keyword

```
class VoterEligility extends Exception
{
   VoterEligility(String s)
   {
      super(s);
   }
}
class UserDefinedExcep
{
   public static void main(String[] args)
   {
      int age = Integer.parseInt(args[0]);
      if(age<18)
         throw new VoterEligility("Not Eligible for vote");
      else
         System.out.println(age+" Eligible for vote");
   }
}
```

Example:

Create a Checked Exception

unreported exception VoterEligility; must be caught or declared to be thrown

throw new VoterEligility("Not Eligible for vote");

User Defined Compile time Exception using try catch

Exception class

```
class VoterEligility extends Exception
{
   String str;
   VoterEligility(String s)
   {
      str = s;
   }
   public String toString()
   {
      return "Not Eligible for vote";
   }
}
```

Class throw Exception

```
class UserDefinedExcep
{
   public static void main(String[] args)
   {
      int age = Integer.parseInt(args[0]);
      try
      {
         if(age<18)
            throw new VoterEligility("Not Eligibleto vote");
         else
            System.out.println(age+" Eligible for vote");
      }
      catch(VoterEligility e)
      {
         System.out.println(e);//call toString method
      }
   }
}
```


Exception Handling with Method Overriding

Superclass Method	Subclass Method
method does not declare with an exception	❖ cannot declare the checked exception ❖ can declare unchecked exception.
superclass method declares an exception	❖ cannot declare parent exception. ❖ can declare same exception ❖ can declare subclass exception or no exception

Exception Handling with Method Overriding

Case 1:

```
import java.io.*;
class Parent{
  void msg()
  {System.out.println("parent");}
}
class TestExceptionChild extends Parent{
  void msg()throws IOException
 {
    System.out.println("TestExceptionChild");
  }
  public static void main(String args[]){
   Parent p=new TestExceptionChild();
   p.msg();
  }
}
```

- superclass method does not declare with an exception.
- subclass overridden method cannot declare the checked exception.

Output:Compile Time Error

Exception Handling with Method Overriding

Case 2:

```
import java.io.*;
class Parent
{
  void msg()
  {
    System.out.println("parent");
  }
}
class TestExceptionChild extends Parent
{
  void msg() throws ArithmeticException
  {
    System.out.println("TestExceptionChild");
  }
  public static void main(String args[])
  {
   Parent p=new TestExceptionChild();
   p.msg();
  }
}
```

- superclass method does not declare an exception.
- subclass overridden method can declare the Unchecked exception.

Output:TestExceptionChild

Exception Handling with Method Overriding

Case 3:

```
import java.io.*;
class Parent{
  void msg() throws ArithmeticException
  {System.out.println("parent");}
}
class TestExceptionChild extends Parent{
  void msg()throws ArithmeticException
 {
    System.out.println("TestExceptionChild");
  }
  public static void main(String args[]){
   Parent p=new TestExceptionChild();
   p.msg();
  }
}
```

- superclass method declare an exception.
- subclass overridden method can declare the same exception,

Output:TestExceptionChild

Exception Handling with Method Overriding

Case 5:

```
import java.io.*;
class Parent{
  void msg() throws Exception
  {System.out.println("parent");}
}
class TestExceptionChild extends Parent{
  void msg()throws ArithmeticException
 {
    System.out.println("TestExceptionChild");
  }
  public static void main(String args[]){
   Parent p=new TestExceptionChild();
   p.msg();
  }
}
```

- superclass method can declare an Base exception.
- subclass overridden method can declares child exception

Output:Compile Time Error

Demo

Quick recap

- try - for fullychecked exception try block must contain that cause exception
- catch – to maintain exception hadling code
- finally – to maintain cleanup code
- throw - used to throw user defined exception
- throws – used to throw checked exception

Quiz

1

What is the output of the following code snippet?

```
class Main {
    public static void main(String args[]) {
      try {
        throw 10;
      }
      catch(int e) {
        System.out.println("Got the  Exception " + e);
      }
   }
 }
```

A. The code does not compile.
B. Got the Exception 0
C. Got the Exception 10
D. Code compile fine but no output

A The code does not compile

2

What is the output of the following code snippet?

```
class Test extends Exception { }

class Main {
  public static void main(String args[]) {
    try {
      throw new Test();
    }
    catch(Test t) {
      System.out.println("Got the Test Exception");
    }
    finally {
      System.out.println("Inside finally block ");
    }
  }
}
```

A. Got the Test Exception
Inside finally block

B. Got the Test Exception

C. Inside finally block

D. Compiler Error

A. Got the Test Exception
Inside finally block

3

What is the output of the following code snippet?

```
class Main {
    public static void main(String args[]) {
        int x = 0;
        int y = 10;
        int z = y/x;
    }
}
```

A. Compiler Error
B. Compiles fine but throws ArithmeticException exception
C. Compiles and runs fine
D. No output

B. Compiles fine but throws ArithmeticException exception

4

What is the output of the following code snippet?

```
class Base extends Exception {}
class Derived extends Base  {}

public class Main {
  public static void main(String args[]) {

    try {
       throw new Derived();
     }
     catch(Base b)     {
       System.out.println("Caught base class exception");
     }
     catch(Derived d)  {
       System.out.println("Caught derived class exception");
     }
   }
}
```

A. Caught base class exception
B. Caught derived class exception
C. Compiler Error because derived is not throwable
D. Compiler Error because base class exception is caught before derived class

D. Compiler Error because base class exception is caught before derived class

5

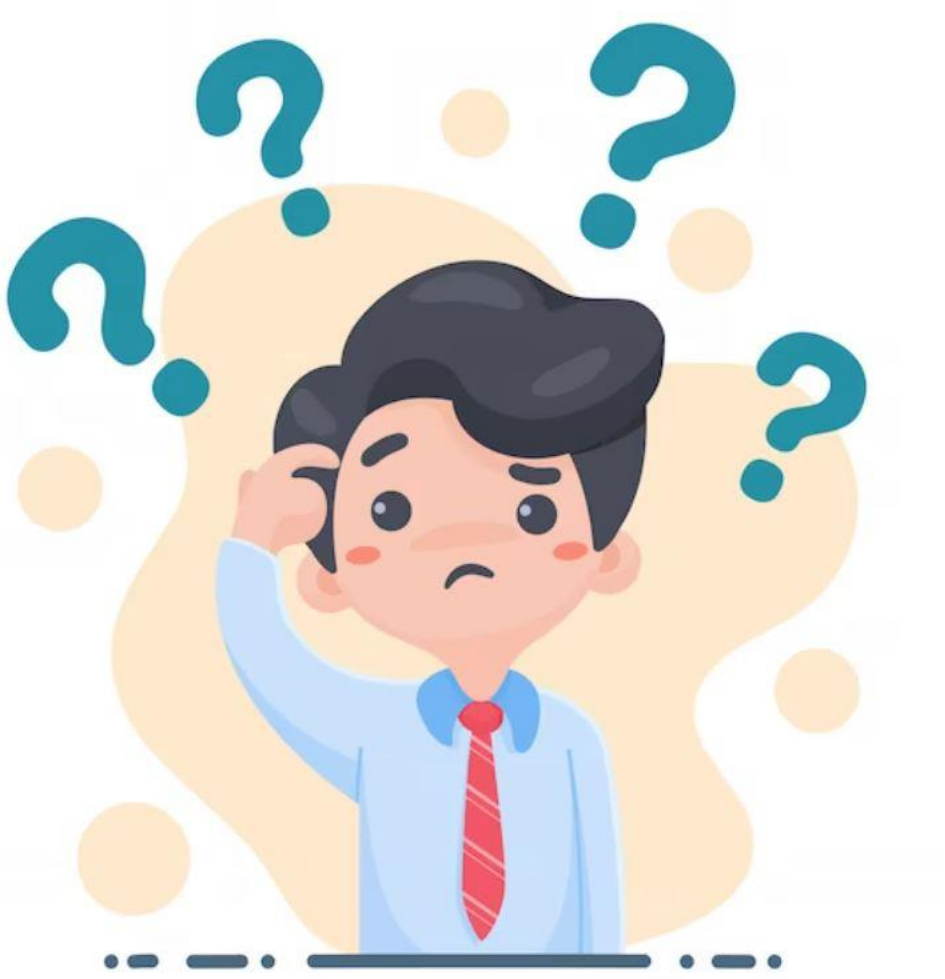
What is the output of the following code snippet?

```
class Test
{
   public static void main(String[] args)
   {
      try
      {
         int a[]= {1, 2, 3, 4};
         for (int i = 1; i <= 4; i++)
         {
            System.out.println ("a[" + i + "]=" + a[i] + "n");
         }
      }
      catch (Exception e)
      {
         System.out.println ("error = " + e);
      }
      catch (ArrayIndexOutOfBoundsException e)
      {
         System.out.println ("ArrayIndexOutOfBoundsException");
      }
   }
}
```

A. Compiler error B. Run time error C. Error Code is printed
D. Array is printed

A. Compiler error

Queries



References

1. https://www.geeksforgeeks.org/java/
2. https://www.tutorialspoint.com/java/index.htm
3. https://www.edureka.co/blog/java-tutorial/
4. https://www.javatpoint.com/java-tutorial
5. https://www.programiz.com/java-programming

Thank you

Innovative Services

Passionate Employees

Delighted Customers