CET1042B: Object Oriented Programming with C++ and Java

SCHOOL OF COMPUTER ENGINEERING AND TECHNOLOGY

S. Y. B. TECH. COMPUTER SCIENCE AND ENGINEERING
(CYBERSECURITY AND FORENSICS)

CET1042B: Object Oriented Programming with C++ and Java

Teaching Scheme
Theory: 2 Hrs. / Week

Credits: 02 + 02 = 04
Practical: 4 Hrs./Week

Course Objectives

1) **Knowledge:** (i) Learn object oriented paradigm and its fundamentals.
2) **Skills:** (i) Understand Inheritance, Polymorphism and dynamic binding using OOP.
   (ii) Study the concepts of Exception Handling and file handling using C++ and Java.
3) **Attitude:** (i) Learn to apply advanced concepts to solve real world problems.

Course Outcomes

1) Apply the basic concepts of Object Oriented Programming to design an application.
2) Make use of Inheritance and Polymorphism to develop real world applications.
3) Apply the concepts of exceptions and file handling to store and retrieve the data.
4) Develop efficient application solutions using advanced concepts.

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Assignment 4

Implementation of exception handling using C++ and JAVA .

MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

EXCEPTION HANDLING Using C++

Laboratory Assignment No: 4

Problem Statement

Define a class Employee consisting following:

Data members:

Employee ID, Name of Employee, Age, Income, City, Vehicle

Member Functions:

To assign initial values.

To display.

Accept Employee ID, Name, Age, Income, City and Vehicle from the user. Create an exception to check the following conditions and throw an exception if the condition does not meet.

1. Employee age between 18 and 55
2. Employee income between Rs. 50,000 – Rs. 1,00,000 per month
3. Employee staying in Pune/ Mumbai/ Bangalore / Chennai
4. Employee having 4-wheeler

Points related to Problem Statement

Exceptions
Exception Handling Mechanism
Try-catch-throw block
Catch all exceptions

Algorithm

1. START.
2. Create an Employee class.
3. Define accept and display member functions of class.
4. Accept Employee Information (Employee ID, Name, Age, Income, City Vehicle).
5. Check the age of an Employee is in between 18 to 55, if not caught an Exception.
6. Check the income of an Employee is in between 50,000 to 1,00,000/-, if not caught an Exception.
7. Check the City of an Employee is Pune/ Mumbai/ Bangalore / Chennai, if not caught an Exception.
8. Check the vehicle of an Employee is 4 wheeler, if not caught an Exception.
9. If none of the exception occurs then display an Employee Information.
10. Else display the corresponding exception caught messages.
11. STOP

Practice Assignments

Develop a program with the following:

a) A function to read two double type numbers from keyboard.

b) A function to calculate the division of these two numbers

c) A try block to throw an exception when a wrong type of data is keyed in

d) A try block to detect and throw an exception if the condition “divide by zero “ occurs.

e) Appropriate catch block to handle the exception thrown.

Exceptions

Exceptions are run time anomalies or unusual condition that a program may encounter during execution.

Examples:

- Division by zero
- Access to an array outside of its bounds
- Running out of memory
- Running out of disk space

Principles of Exception Handling

Similar to errors, exceptions are also of two types. They are as follows:

- **Synchronous exceptions**: The exceptions which occur during the program execution due to some fault in the input data.

For example: Errors such as out of range, overflow, division by zero

- **Asynchronous exceptions**: The exceptions caused by events or faults unrelated (external) to the program and beyond the control of the program.

For Example: Key board failures, hardware disk failures

Exception Handling Mechanism

Exception handling mechanism basically builds upon three keywords:

- try
- catch
- throw

The keyword **try** is used to preface a block of statements which may generate exceptions.

Syntax of try statement:

```
try
{
        statement 1;
        statement 2;
}
```

Throw statement

When an exception is detected, it is thrown using a **throw** statement in the try block.

Syntax of throw statement

- throw (excep);
- throw excep;
- throw; // re-throwing of an exception

Catch block

A **catch** block defined by the keyword 'catch' catches the exception and handles it appropriately. The catch block that catches an exception must immediately follow the try block that throws the exception

Syntax of catch statement: try

```
{
Statement 1;
Statement 2;
}
catch ( argument)
{
statement 3; // Action to be taken
}
```

When an exception is found, the catch block is executed. The catch statement contains an argument of exception type, and it is optional. When an argument is declared, the argument can be used in the catch block. After the execution of the catch block, thestatements inside the blocks are executed. In case no exception is caught, the catch block is ignored, and if a mismatch is found, the program is terminated.

Example:

Write a program to illustrate division by zero exception. */

```
#include <iostream>  using namespace std;

int main()
{
int a, b;
cout<<"Enter the values of a and b"<<endl;  cin>>a>>b;
try{
if(b!=0)
cout<<a/b;
else
throw b;
}
catch(int i)

        {
cout<<"Division by zero: "<<i<<endl;
}

return 0;
}
```

Output:

```
Enter the values of a and b  2
0

Division by zero: 0
```

Multiple Catch

It is also possible that a program segment has more than one condition to throw an exception. In such cases, we can associate more than one catch statement with a try (similar to switch statement). The format of multiple catch statements is as follows:

Syntax:

```
try
{
// try block
}
catch (type1 arg)
{
// catch section1
}
catch (type2 arg)
{
// catch section2
}
. . . . . .
. . . . . .
catch (typen arg)
{
// catch section-n
}
```

Example

/*Write a C++ program to throw multiple exceptions and define multiple catch statement. */

```
#include <iostream>
 using namespace std;

void num (int k)
    {
 try
{
if (k==0) throw k;  else
if (k>0) throw 'P';
else
if (k<0) throw 1.0;  cout<<"*** end of try block ***\n";
}
catch(char g)
{
cout<<"Caught a positive value \n";
}
catch (int j)
{
cout<<"caught an null value \n";
}
catch (double f)
{
cout<<"Caught a Negative value \n";
}
cout<<"*** end of try catch ***\n\n;
}
int main()
{
cout<<"Demo of Multiple catches"<<endl;  num(0);
num(5);
num(-1);  return 0;
}
```

Output:

```
Demo of Multiple catches  caught an null value
*** end of try catch ***  Caught a positive value
*** end of try catch ***  Caught a Negative value
*** end of try catch ***  Caught a positive value
*** end of try catch ***
```

Example

/* Write a C++ program to restrict a function to `throw` only specified type of exceptions.*/

```
#include<iostream>
using namespace std;

void check (int k) throw (int)
{
if (k==1) throw 'k';  else
if (k==2) throw k;
else
if (k==-2) throw 1.0;
}
```

```
int main()
{
try {
check(1);
check(-2);
        check(3);
       }
        catch (char g)
       {
cout<<"Caught a character exception \n";
       }
       catch (int j)
       {
cout<<"Caught a character exception \n";
       }
       catch (double s)
       {
cout<<"Caught a double exception \n";
       }
cout<<"\n End of main()";  return 0;
}
```

MIT-WPU

॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

EXCEPTION HANDLING USING JAVA

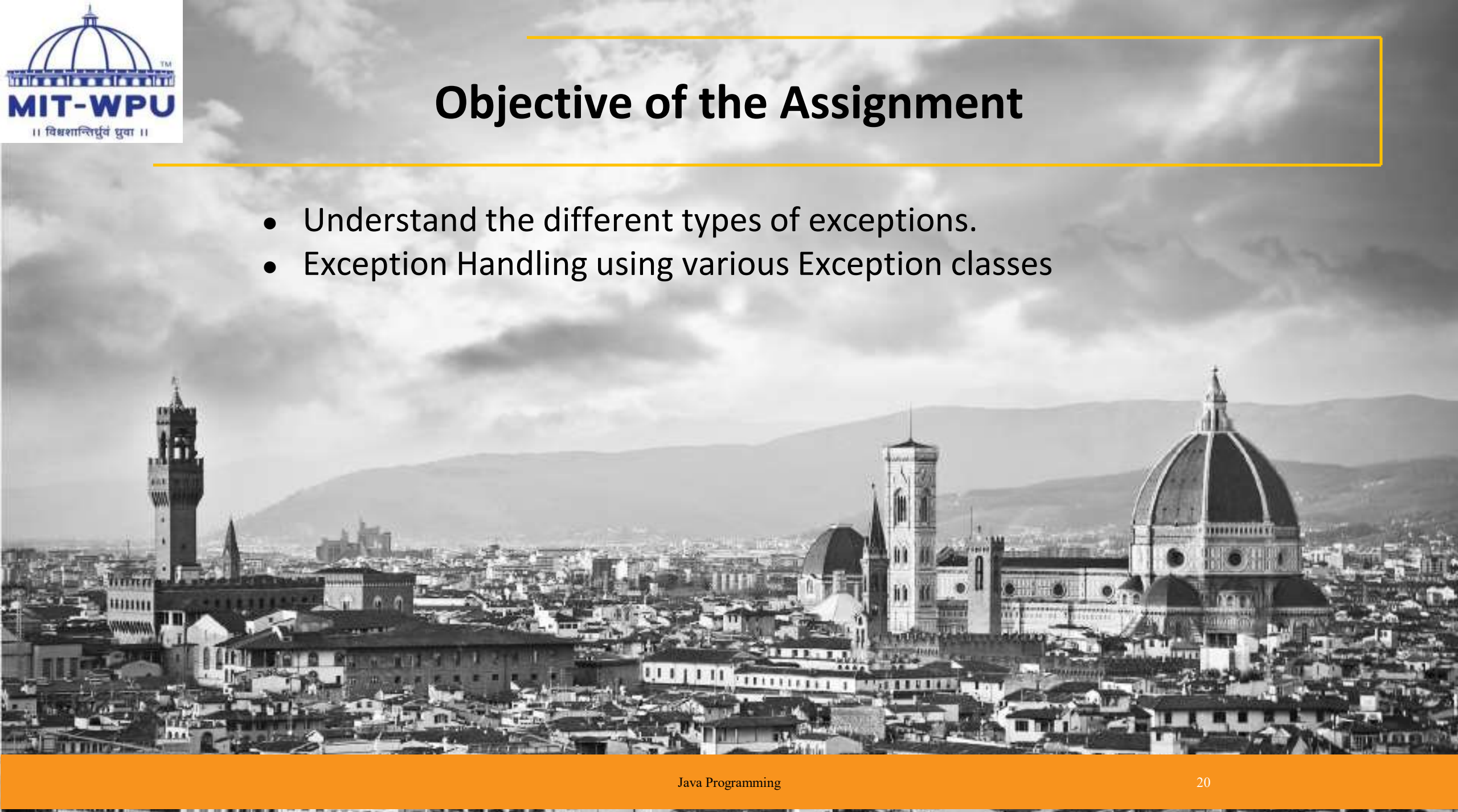
Problem Statement

Write a Java program to showcase the use of Exception Handling in Java

-Exception Handling using various Exception classes

Objective of the Assignment

- Understand the different types of exceptions.
- Exception Handling using various Exception classes



Problem 1

A. Implement the program to handle the arithmetic exception , ArrayIndexOutofBounds . The user enters the two numbers: n1,n2. The division of n1 and n2 is displayed. If n1, n2 are not integers then program will throw number format exception. If n2 is zero the program will throw Arithmetic exception.

B. Validate the employee record with custom exception

Create a class employee with attributes eid, name, age and department.

Initialize values through parameterized constructor. If age of employee is not in between 25 and 60 then generate user-defined exception "AgeNotWithinRangeException". If name contains numbers or special symbols raise exception "NameNotValidException". Define the two exception classes.

Problem 2

Write a menu-driven program for banking system which accept the personal data for Customer(cid, cname, amount).
Implement the user-defined/standard exceptions, wherever required to handle the following situations:

- Account should be created with minimum amount of 1000 rs..
- For withdrawal of amount, if wth_amt>amount.
- cid should be in the specific range of 1 to 20.
- Entered amount should be positive.

Exception Handling in Java

- An exception is an unexpected event that occurs during program execution.
- It affects the flow of the program instructions which can cause the program to terminate abnormally.
- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.
- **All the exceptions occur only at runtime.**
- **A syntax error occurs at compile time.**
- An exception can be identified only at runtime, not at compile time.

Examples of Exceptions

- Opening a non-existing file in your program.
- Reading a file from a disk but the file does not exist there.
- Writing data to a disk but the disk is full or unformatted.
- When the program asks for user input and the user enters invalid data.
- When a user attempts to divide an integer value by zero, an exception occurs.
- When accessing an array index where value does not exist, etc.

Exception Class Hierarchy

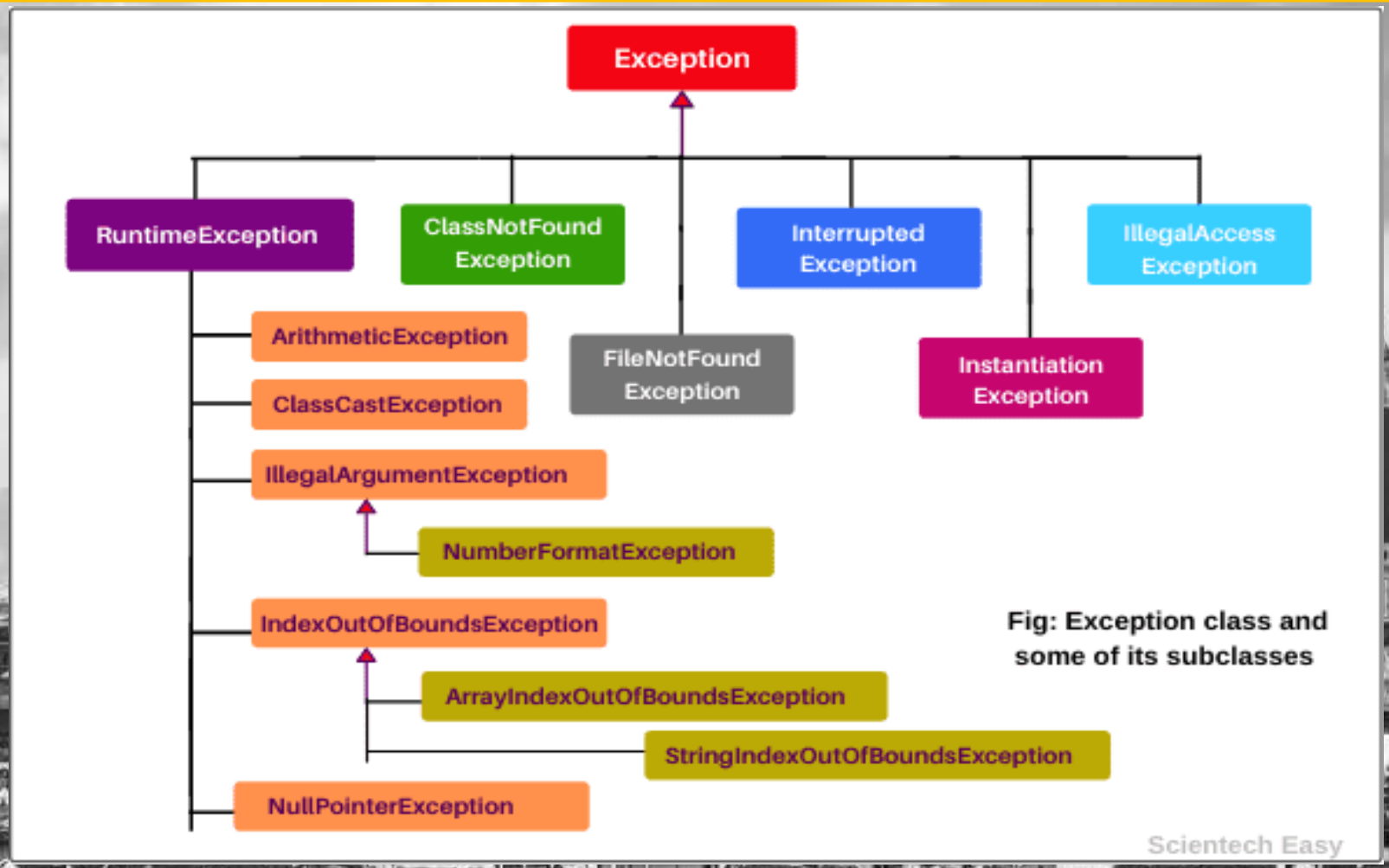
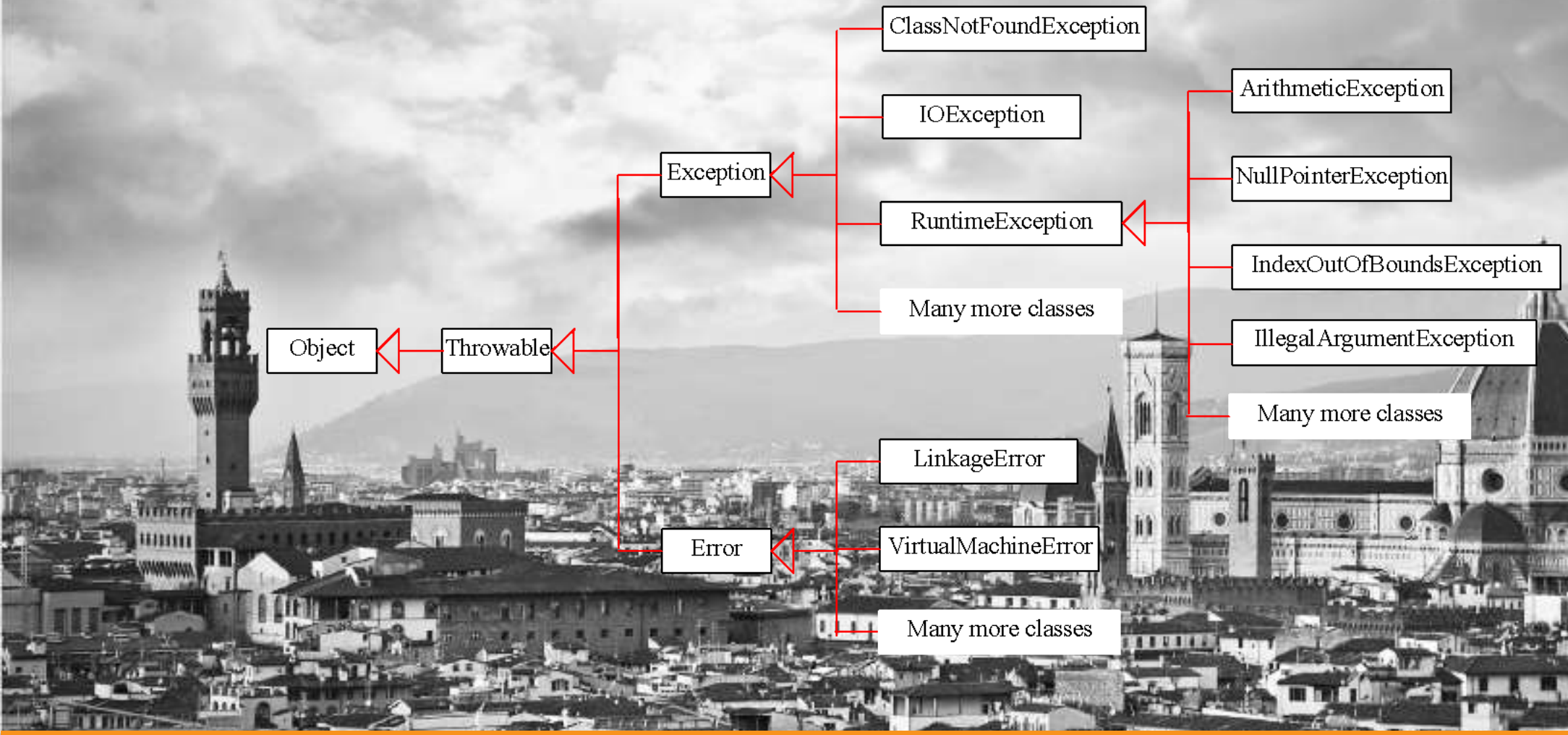


Fig: Exception class and some of its subclasses

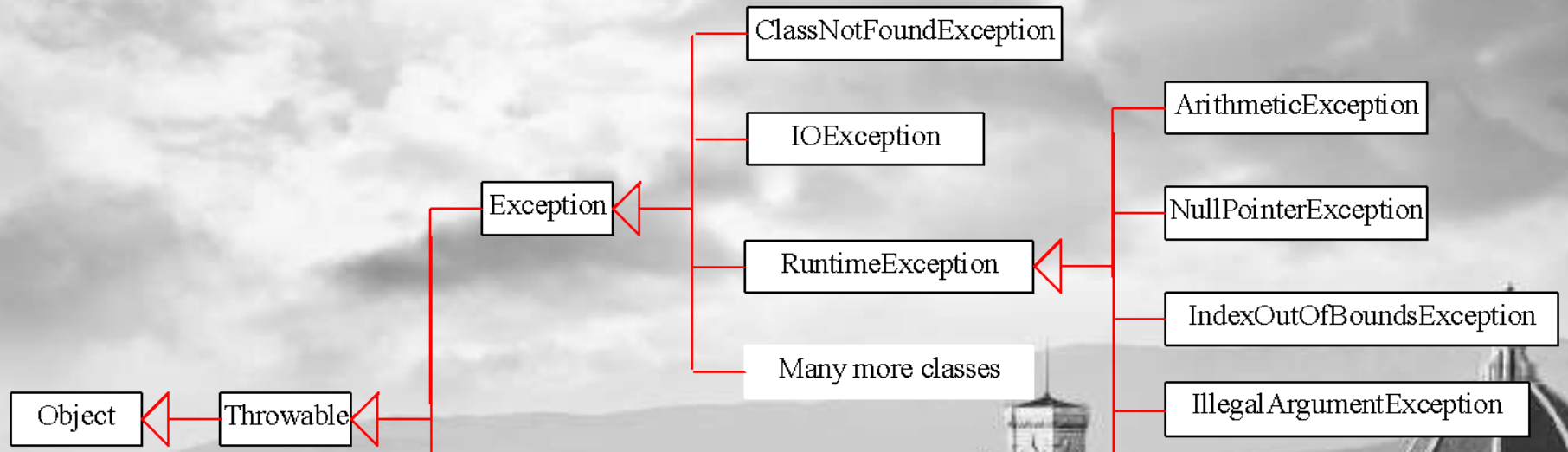
Error Vs Exception

Type of comparison	Exception	Error
Package	It belongs to java.lang.Exception package	It belongs to java.lang.Error package.
Occurrence	It occurs at compile time or run time.	It occurs at run time.
Causes	It is mainly caused by the application itself.	It is mostly caused by the environment in which the application is running.
Recoverable/ Irrecoverable	It is recoverable	It is irrecoverable
Type	Classified as an unchecked type	Classified as checked and unchecked
Example	Checked Exception- NullPointerException , SqlException Uncheckedexceptions: ArrayIndexOutOfBoundException, NullPointerException, ArithmaticException	OutOfMemoryError ,IOError

Exception Types



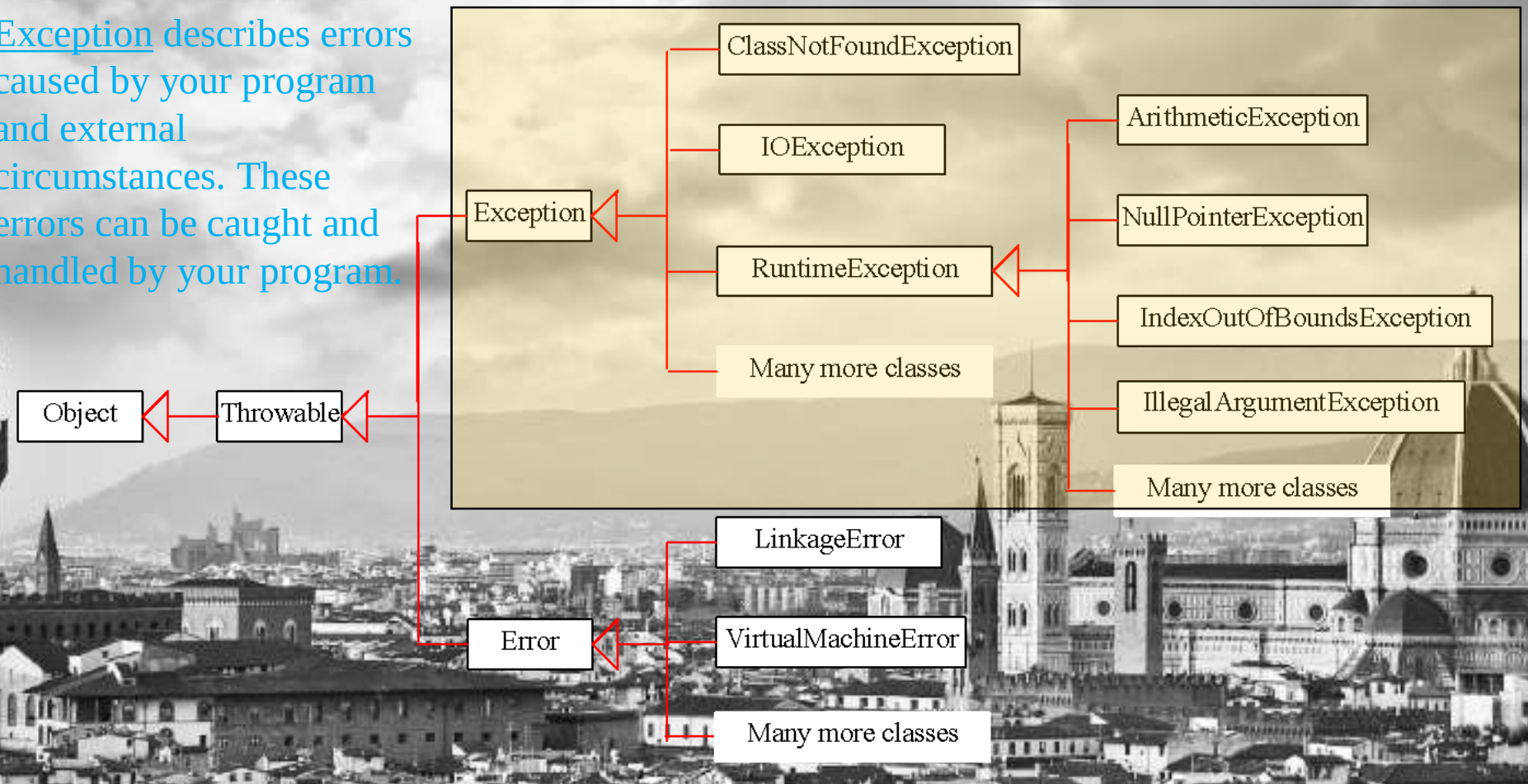
System Errors



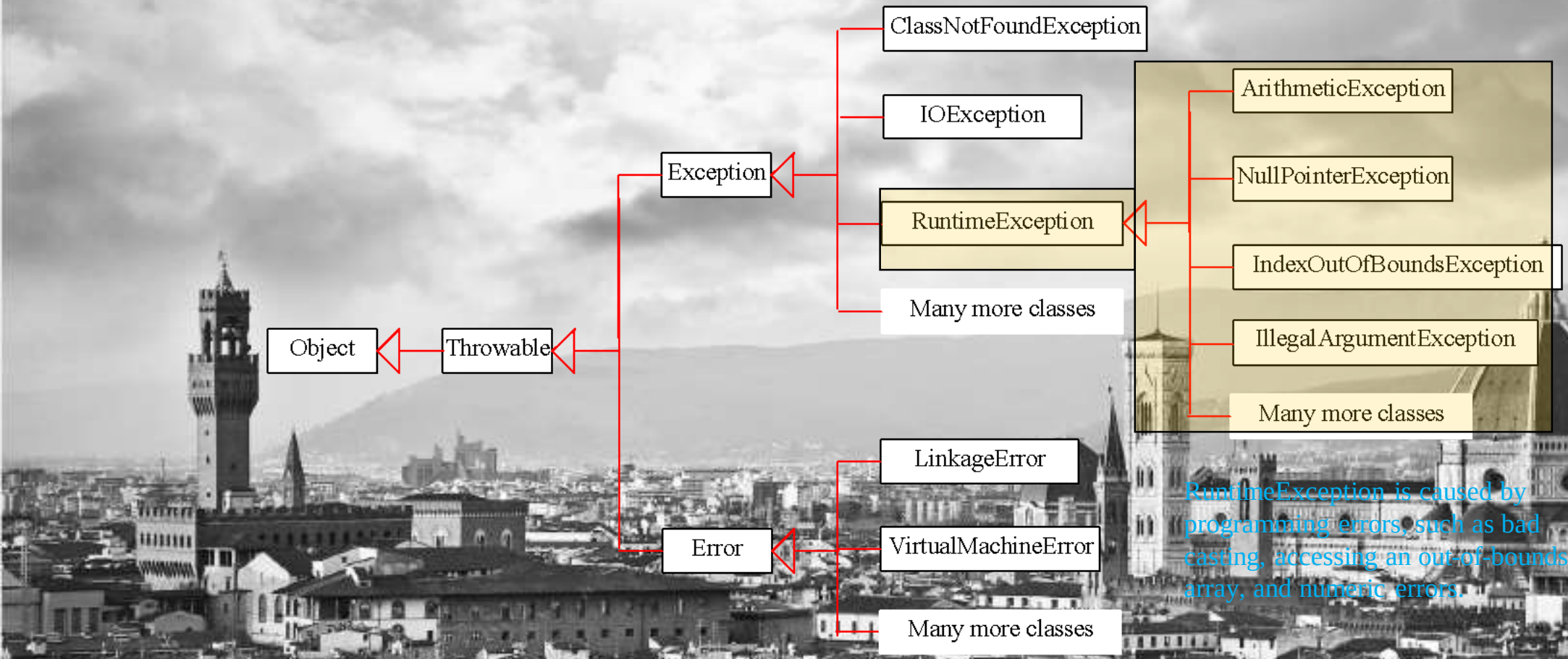
System errors are thrown by JVM and represented in the Error class. The Error class describes internal system errors. Such errors rarely occur. If one does, there is little you can do beyond notifying the user and trying to terminate the program gracefully.

Exceptions

Exception describes errors caused by your program and external circumstances. These errors can be caught and handled by your program.



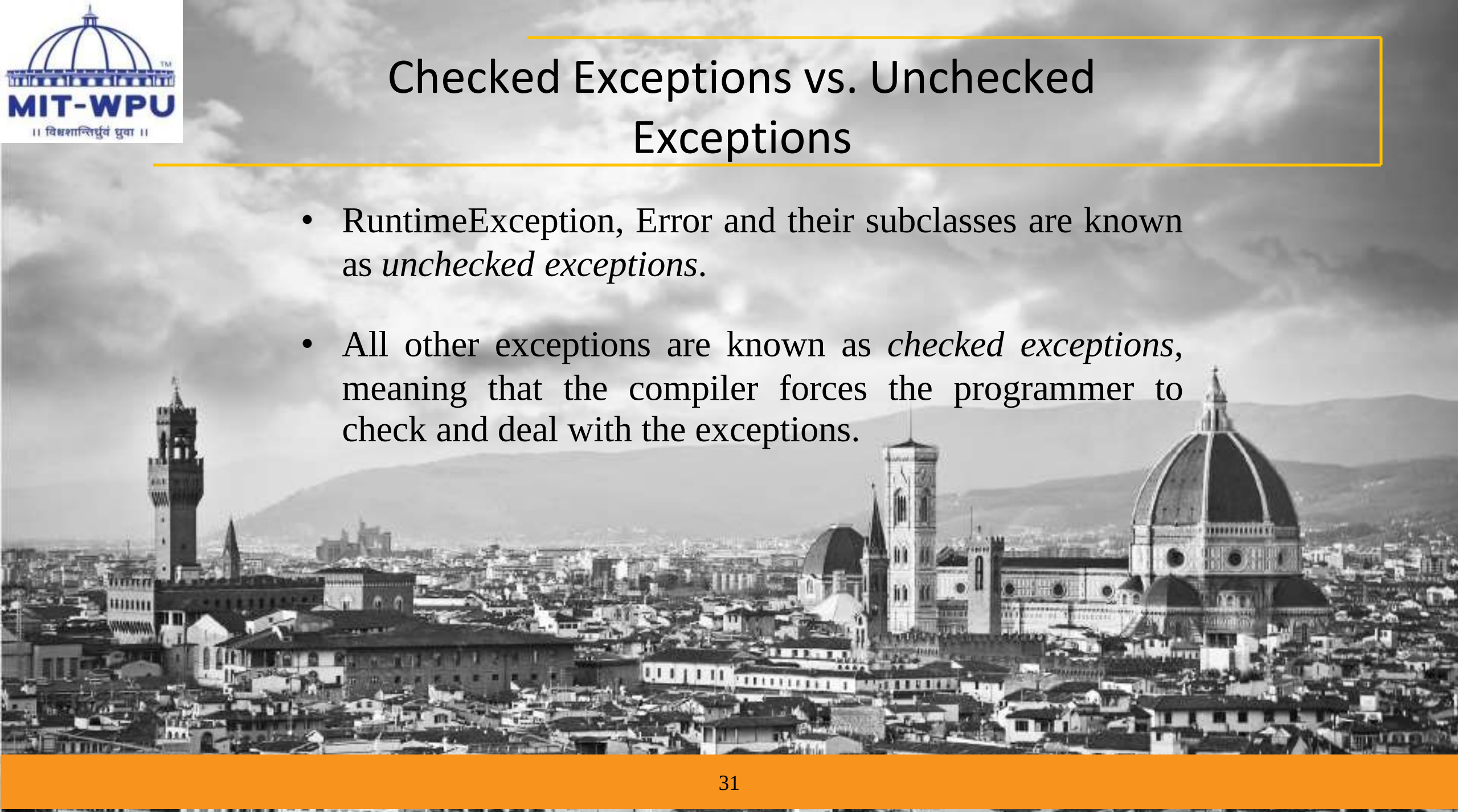
Runtime Exceptions



RuntimeException is caused by programming errors, such as bad casting, accessing an out-of-bounds array, and numeric errors.

Checked Exceptions vs. Unchecked Exceptions

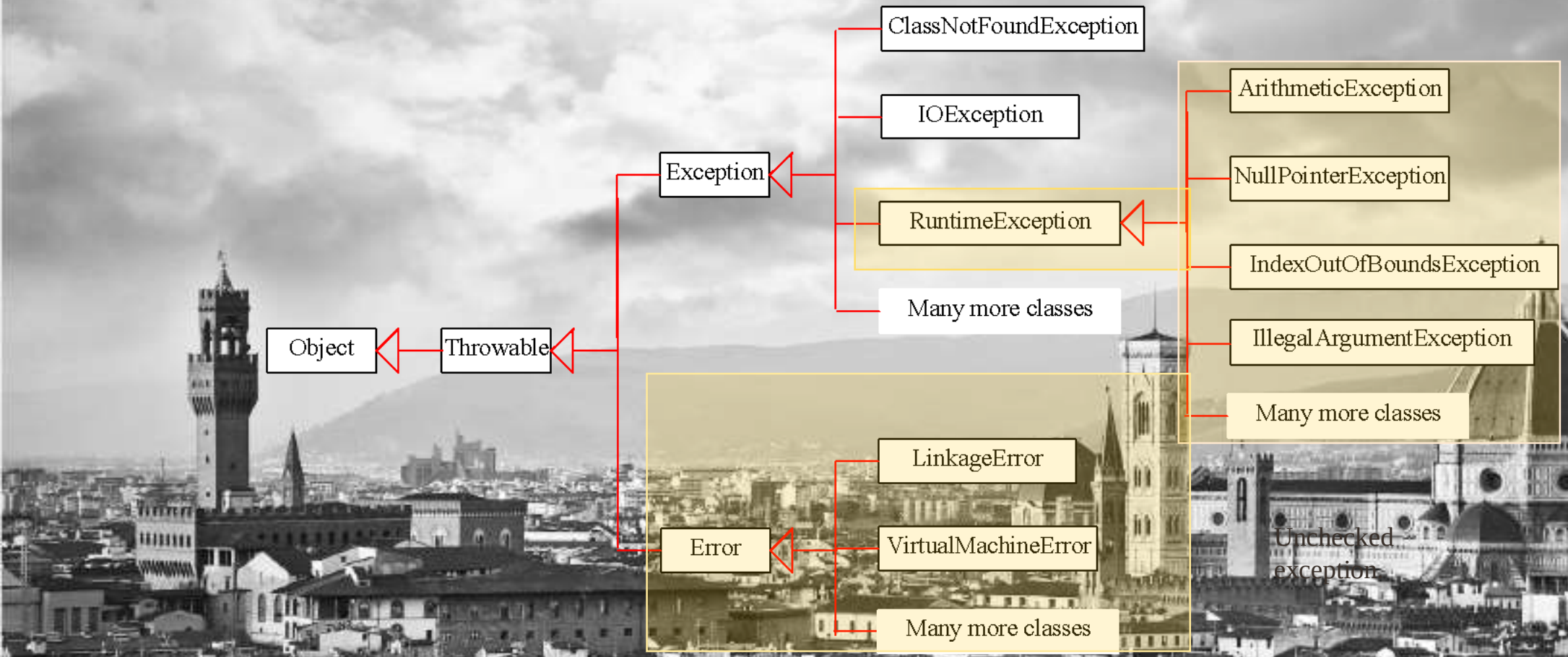
- RuntimeException, Error and their subclasses are known as *unchecked exceptions*.
- All other exceptions are known as *checked exceptions*, meaning that the compiler forces the programmer to check and deal with the exceptions.



Unchecked Exceptions

- In most cases, unchecked exceptions reflect programming logic errors that are not recoverable. For example, a NullPointerException is thrown if you access an object through a reference variable before an object is assigned to it;
- An IndexOutOfBoundsException is thrown if you access an element in an array outside the bounds of the array.
- Unchecked exceptions can occur anywhere in the program. To avoid cumbersome overuse of try-catch blocks, Java does not mandate you to write code to catch unchecked exceptions.

Unchecked Exceptions



Exception handling mechanism

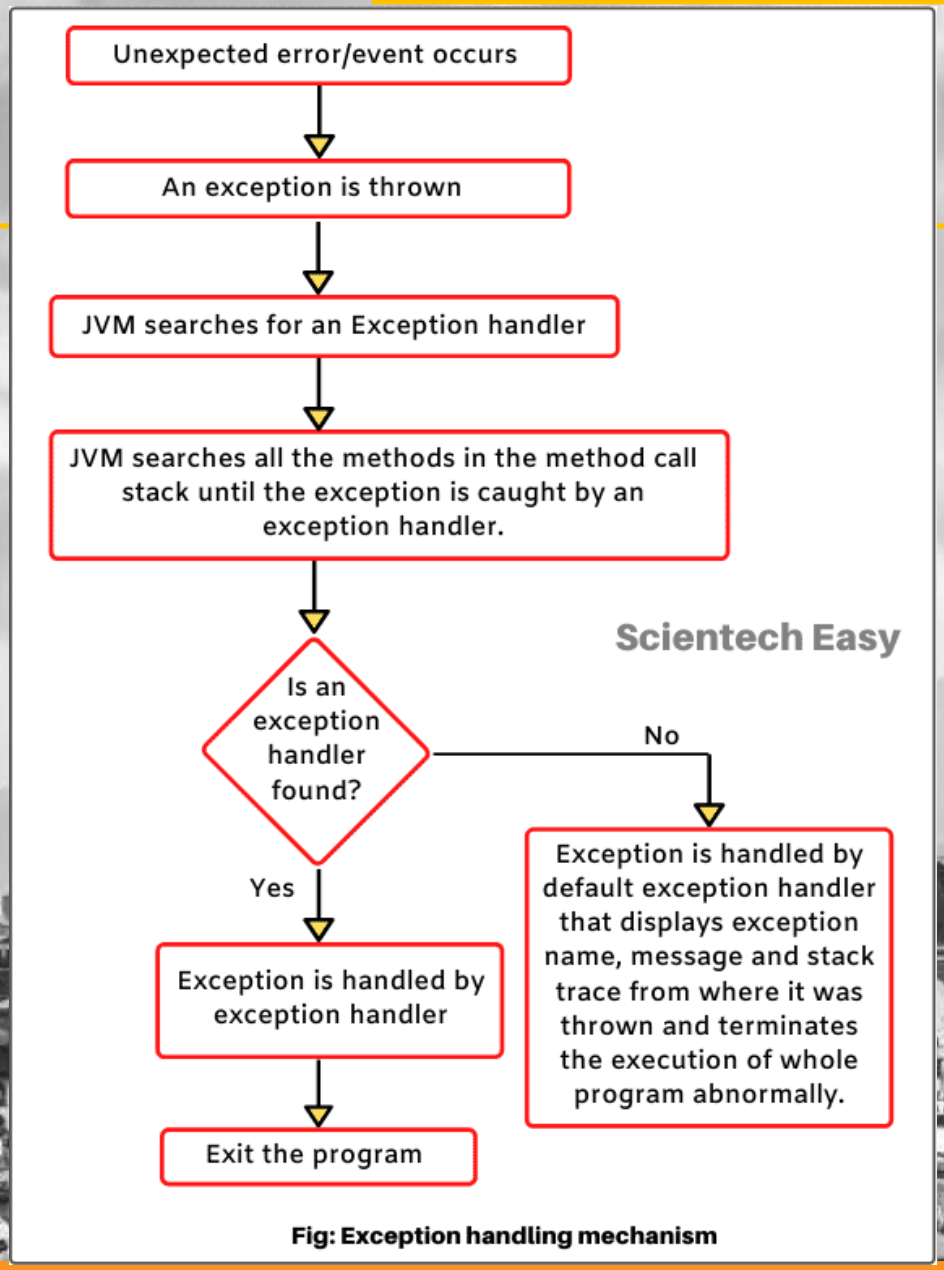


Fig: Exception handling mechanism

Exception Handling using Try-Catch

- The try block contains set of statements where an exception can occur.
- A try block is always followed by a catch block, which handles the exception that occurs in associated try block.
- A try block must be followed by catch blocks or finally block or both.

Syntax of try *block*

```
try
{
    //statements that may cause an exception
}
```

While writing a program, if you think that certain statements in a program can throw a exception, enclosed them in try block and handle that exception

Catch Block

- A catch block is where you handle the exceptions, this block must follow the try block.
- A single try block can have several catch blocks associated with it.
- You can catch different exceptions in different catch blocks.
- When an exception occurs in try block, the corresponding catch block that handles that particular exception executes.
- For exampleif an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.

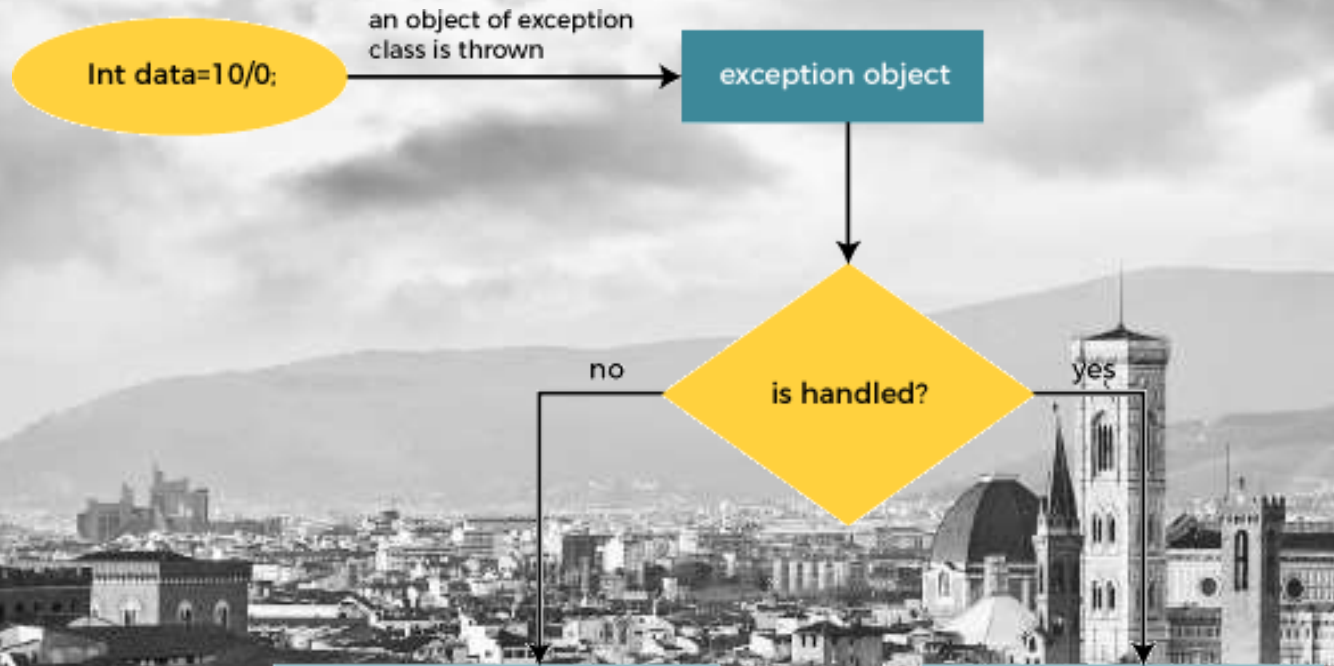
Try-Catch Example

Syntax of try catch in java

```
try
{
//statements that may cause an exception
}
catch (exception(type) e(object))
{
//error handling code
}
```

```
public class re {

    public static void main(String[] args) {
        try
        {
        int num = 45/0; // throw exception

            //handling the exception
        catch(ArithmeticException e)
        {
        System.out.println("exception->"+e);
        }

    System.out.println("rest of the code");
    }

}
```

Internal Working



Multiple Catch Block

1. A single try block can have any number of catch blocks.
2. A generic catch block can handle all the exceptions. Whether it is ArrayIndexOutOfBoundsException or ArithmeticException or NullPointerException or any other type of exception, this handles all of them.
3. If no exception occurs in try block then the catch blocks are completely ignored.
4. Corresponding catch blocks execute for that specific type of exception:

catch(ArithmeticException e) is a catch block that can handle ArithmeticException

catch(NullPointerException e) is a catch block that can handle NullPointerException

```
public class re {

    public static void main(String[] args)
{
     Scanner sc=new Scanner(System.in);
          System.out.println("Enter two
integer numbers");
// Read two integer numbers.
  int num1 = sc.nextInt();
  int num2 = sc.nextInt();
     try {
 System.out.println(num1 + "/" + num2 + "
= " + (num1/num2));
        }catch(ArithmeticException e)
//multiple catch statement
   {
          System.out.println("Divide by 0");
        }catch(Exception e) //superClass
   {
        System.out.println("divide by 0");
        }finally {  //finally block gets
executed regardless of try... catch.
System.out.println("The 'try catch' is
finished.");
        }
}}
```

Multiple Catch Block-Example

```
public class re {

    public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
        System.out.println("Enter two integer numbers");
// Read two integer numbers.
  int num1 = sc.nextInt();
  int num2 = sc.nextInt();
    try {
 System.out.println(num1 + "/" + num2 + " = " + (num1/num2));
      }catch(ArithmeticException e)    //multiple catch statement
  {
            System.out.println("Divide by 0");
      }catch(Exception e) //superClass
  {
          System.out.println("divide by 0");
          }finally {    //finally block gets executed regardless of try...
catch.
            System.out.println("The 'try catch' is finished.");
      }
  }}
```

Nested Try-Catch

- When a **try catch block** is present in another try block then it is called the nested try catch block.
- Each time a try block does not have a catch handler for a particular **exception**, then the catch blocks of parent try block are inspected for that exception, if match is found then that catch block executes.

```
..
//main try block
try{
  stmt1;
  stmt2;
  //try-catch block inside another block
   try{
       stmt3;
       stmt4;
      }
 //try-catch block inside nested try block
   try{
       stmt5;
       stmt6;
       } catch (Exception e2)
          {  //exception message }
  }catch(Exception e1)
  {//exception message}
}
// catch of main (parent) try block
catch(Exception e3)
{//exception message}
....
```

Finally block

- A finally block must be associated with a try block, you cannot use finally without a try block. You should place those statements in this block that must be executed always.
- Finally block is optional, as we have seen that a try-catch block is sufficient for exception handling, however if you place a finally block then it will always run after the execution of try block.
- In normal case when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.
- An exception in the finally block, behaves exactly like any other exception.
- The statements present in the finally block execute even if the try block contains control transfer statements like return, break or continue.

finally block without catch

- A try-finally block is possible without catch block Which means a try block can be used with finally without having a catch block.

...

```
InputStream input = null;
try {
input = new FileInputStream("inputfile.txt");
}
finally {
if (input != null)
{  try {
in.close();
}catch (IOException exp)
{  System.out.println(exp); }        }    }
```

...

Try-catch-finally block

- Either a try statement should be associated with a catch block or with finally.
- Since catch performs exception handling and finally performs the cleanup, the best approach is to use both of them.

```
try
{
//statements that may cause an exception
}
catch (...)
{
//error handling code
}
Finally
{
//statements to be executed
}
```

Flow of control in try-catch-finally block

1. If exception occurs in try block's body the control immediately transferred(**skipping rest of the statements in try block**) to the catch block. Once catch block finished execution then **finally block** and after that rest of the program.
2. If there is no exception occurred in the code which is present in try block then first, the try block gets executed completely and then control gets transferred to finally block (**skipping catch blocks**).
3. If a **return statement** is encountered either in try or catch block. In this case **finally block runs**. Control first jumps to finally and then it returned back to **return statement.**

Throw in Java

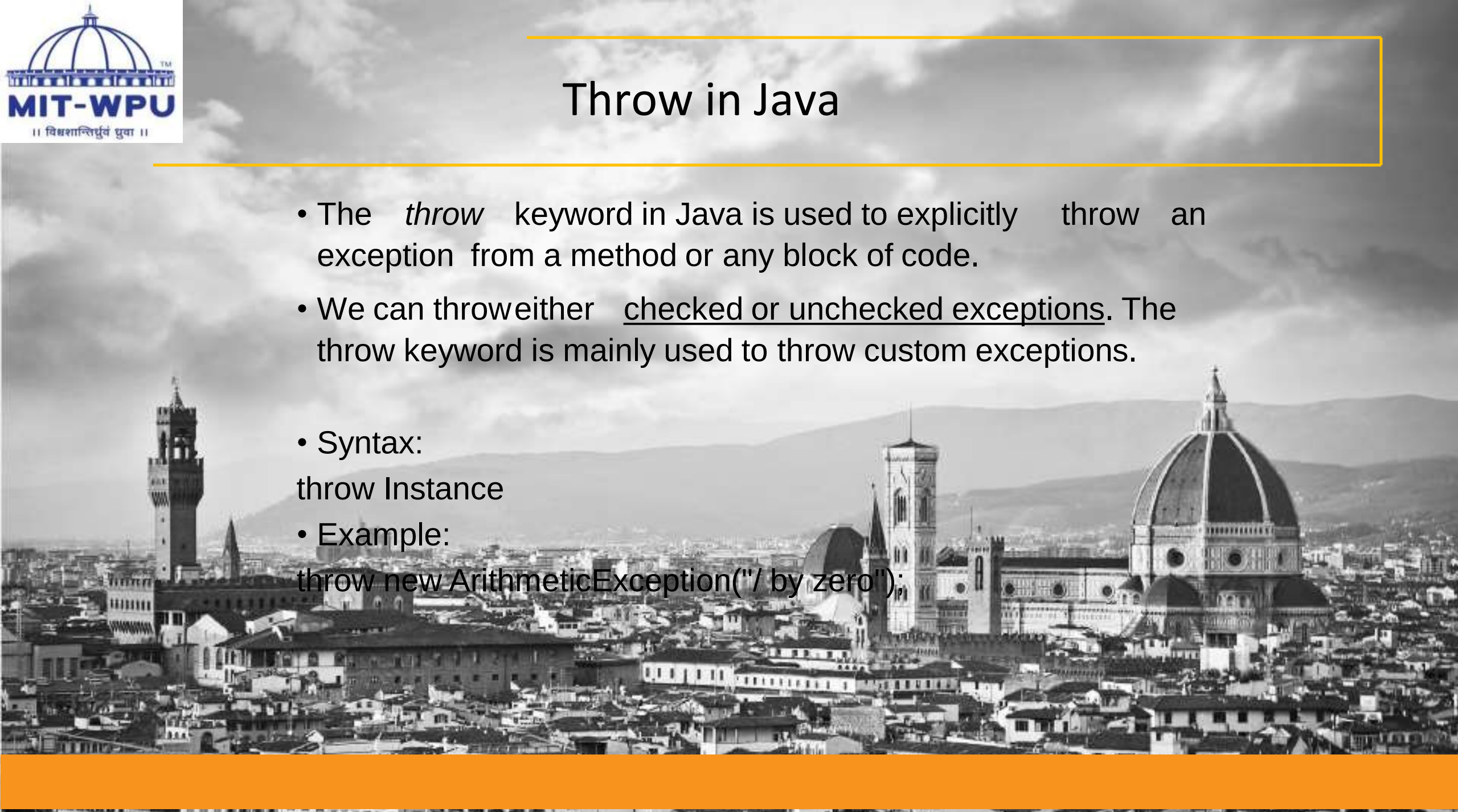
- The *throw* keyword in Java is used to explicitly throw an exception from a method or any block of code.
- We can throweither checked or unchecked exceptions. The throw keyword is mainly used to throw custom exceptions.

- Syntax:

throw Instance

- Example:

throw new ArithmeticException("/ by zero");



Throw in Java

- The flow of execution of the program stops immediately after the throw statement is executed and the nearest enclosing **try** block is checked to see if it has a **catch** statement that matches the type of exception.
- If it finds a match, control is transferred to that statement otherwise next enclosing **try** block is checked and so on.
- If no matching **catch** is found then the default exception handler will halt the program.

Throwing unchecked Exception

Ex. 1

```
class Exception2{
static int sum(int num1, int num2)
  {  if (num1 == 0)
throw new ArithmeticException("First parameter is not valid");
else
System.out.println("Both parameters are correct!!");
return num1+num2;
}
public static void main(String args[])
{  int res=sum(0,12);
   System.out.println(res);
System.out.println("Continue Next statements");
}
}
```

Throwing unchecked Exception

Ex. 2

```
public class test
{
public static void check(int age){
if (age<18)
{ throw new ArithmeticException("not eligible to vote");}
else
{ System.out.println("Person is eligible to vote");
}
}

public static void main(String args[])
{
check(12);
System.out.println("rest of the code");
}
}
```

Throws in Java

- The throws keyword in Java is used to declare exceptions that can occur during the execution of a program.
- For any method that can throw exceptions, it is mandatory to use the throws keyword to list the exceptions that can be thrown.
- The throws keyword provides information about the exceptions to the programmer as well as to the caller of the method that throws the exceptions.

Throws in Java

```
public class re{
public static void checkage(int age) throws ArithmeticException{
if (age<18)
{
throw new ArithmeticException(" not eligible to vote");
}
else
{
System.out.println("Person is eligible to vote");}
}

public static void main(String args[])
{
checkage(21);
}
}
```

Throwing Checked Exception

```
import java.io.*;
public class TestThrow2 {
//function to check if person is eligible to vote or not
public static void method() throws FileNotFoundException {  FileReader file = new
FileReader("C:\\Users\\Desktop\\abc.txt");
BufferedReader fileInput = new BufferedReader(file);

throw new FileNotFoundException();
        }

        //main method
        public static void main(String args[]){  try
        {
        method();
        }
        catch (FileNotFoundException e)
        {
        e.printStackTrace();
        }
        System.out.println("rest of the code...");
                        }
                                }
```

Throw and Throws in Java

Throw	Throws
Used within a method (or constructor)	Used with method (or constructor) signature
Used to throw an exception explicitly	Used to declare exceptions
Can only throw a single exception	Can declare multiple exceptions
Followed by a throwable instance	Followed by an exception class name

User defined exception in Java

In order to create custom exception, we need to extend Exception class that belongs to java.lang package.

```
//class that uses custom exception InvalidAgeException
public class re  {
 // method to check the age
 static void validate (int age) throws Exception{
    if(age < 18){
     // throw an object of user defined exception
     throw new Exception("age is not valid to vote");
 }
    else {
     System.out.println("welcome to vote");
     }     }
 // main method
 public static void main(String args[])    {
     try        {
          // calling the method
          validate(13);          }
     catch (Exception ex)        {
          System.out.println("Caught the exception");
          // printing the message from InvalidAgeException object
          System.out.println("Exception occured: " + ex);          }
     System.out.println("rest of the code...");
 }  }
```

User defined exception in Java

In order to create custom exception, we need to extend Exception class that belongs to java.lang package.

```
class InvalidAgeException  extends Exception{
    public InvalidAgeException (String str)
    {
       super(str);    // calling the constructor of parent Exception
    }
}
//class that uses custom exception InvalidAgeException
public class re  {
 // method to check the age
 static void validate (int age) throws Exception{
    if(age < 18){
     // throw an object of user defined exception
     throw new InvalidAgeException("age is not valid to vote");
}else {
     System.out.println("welcome to vote");
     }    }
 // main method
 public static void main(String args[])    {
     try        {
                validate(13); // calling the method
     } catch (Exception ex)        {
          System.out.println("Caught the exception");
          // printing the message from InvalidAgeException object
          System.out.println("Exception occured: " + ex);        }
     System.out.println("rest of the code...");
 }  }
```

Categories of Predefined Exceptions

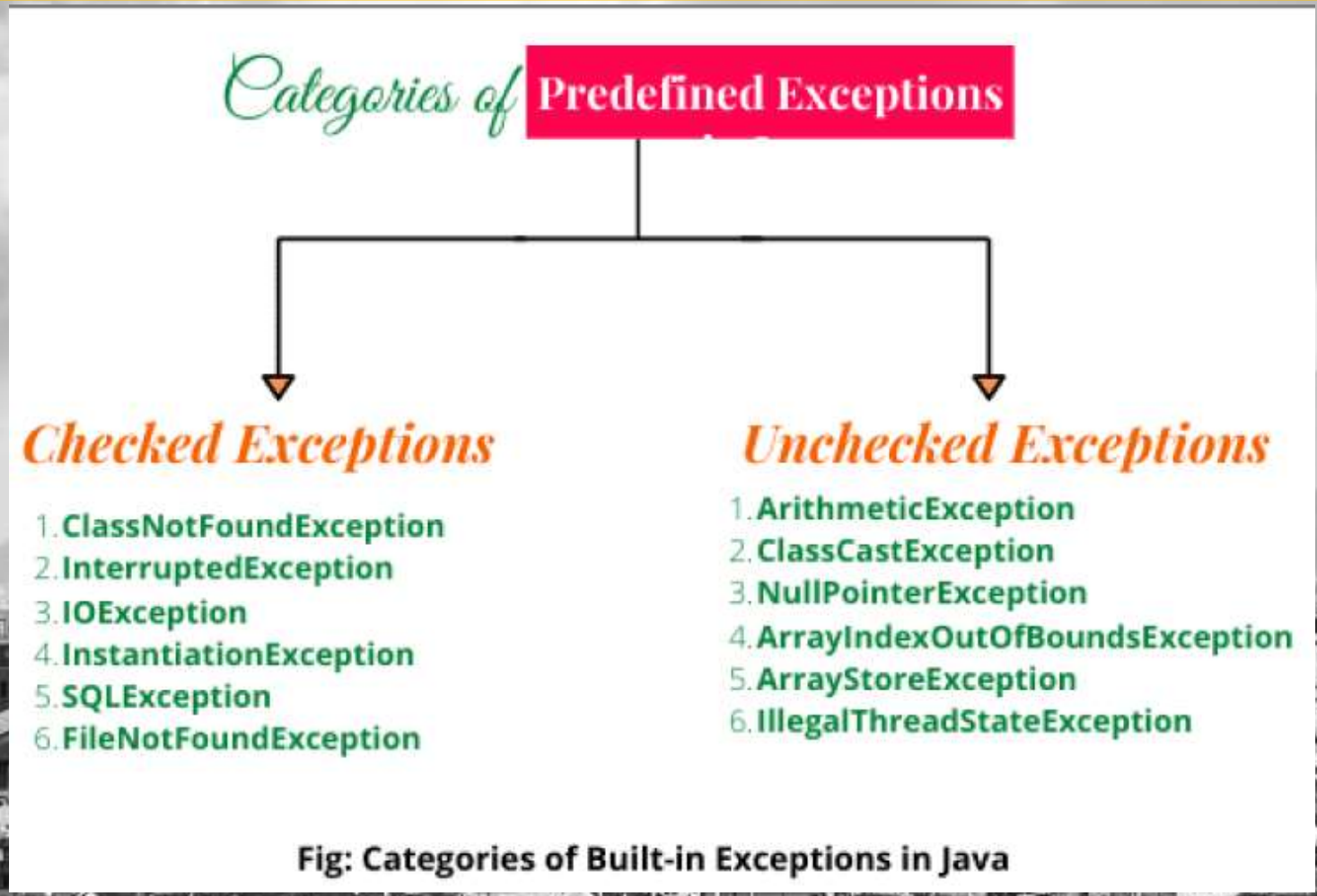


Fig: Categories of Built-in Exceptions in Java

Unchecked Exceptions

1. **ArithmeticException:** This exception is thrown when arithmetic problems, such as a number is divided by zero, is occurred. That is, it is caused by maths error.

2. **ClassCastException:** The ClassCastException is a runtime exception that is thrown by JVM when we attempt to invalid typecasting in the program. That is, it is thrown when we cast an object to a subclass of which an object is not an instance.

3. **IllegalArgumentException:** This runtime exception is thrown by programmatically when an illegal or appropriate argument is passed to call a method. This exception class has further two subclasses:

   - **NumericFormatException:**NumberFormatException is thrown by programmatically when we try to convert a string into the numeric type and the process of illegal conversion fails. That is, it occurs due to the illegal conversion of a string to a numeric format.

   - **IllegalThreadStateException:** IllegalThreadStateException exception is a runtime exception that is thrown by programmatically when we attempt to perform any operation on a thread but it is incompatible with the current thread state.

Unchecked Exceptions

4. IndexOutOfBoundsException: This exception class is thrown by JVM when an array or string is going out of the specified index. It has two further subclasses:

- **ArrayIndexOutOfBoundsException:**

ArrayIndexOutOfBoundsException exception is thrown when an array element is accessed out of the index.

- **StringIndexOutOfBoundsException:**

StringIndexOutOfBoundsException exception is thrown when a String or StringBuffer element is accessed out of the index.

Unchecked Exceptions

5. **NullPointerException:** NullPointerException is a runtime exception that is thrown by JVM when we attempt to use null instead of an object. That is, it is thrown when the reference is null.

6. **ArrayStoreException:** This exception occurs when we attempt to store any value in an array which is not of array type. For example, suppose, an array is of integer type but we are trying to store a value of an element of another type.

7. **IllegalStateException:** The IllegalStateException exception is thrown by programmatically when the runtime environment is not in an appropriate state for calling any method.

8. **IllegalMonitorStateException:** This exception is thrown when a thread does not have the right to monitor an object and tries to access wait(), notify(), and notifyAll() methods of the object.

9. **NegativeArraySizeException:** The NegativeArraySizeException exception is thrown when an array is created with a negative size.

checked Exceptions

1. **ClassNotFoundException:** The ClassNotFoundException is a kind of checked exception that is thrown when we attempt to use a class that does not exist. Checked exceptions are those exceptions that are checked by the Java compiler itself.

2. **FileNotFoundException:** The FileNotFoundException is a checked exception that is thrown when we attempt to access a non-existing file.

3. **InterruptedException:** InterruptedException is a checked exception that is thrown when a thread is in sleeping or waiting state and another thread attempt to interrupt it.

4. **InstantiationException:** This exception is also a checked exception that is thrown when we try to create an object of abstract class or interface. That is, InstantiationException exception occurs when an abstract class or interface is instantiated.

checked Exceptions

5. IllegalAccessException: The IllegalAccessException is a checked exception and it is thrown when a method is called in another method or class but the calling method or class does not have permission to access that method.

6. CloneNotSupportedException: This checked exception is thrown when we try to clone an object without implementing the cloneable interface.

7. NoSuchFieldException: This is a checked exception that is thrown when an unknown variable is used in a program.

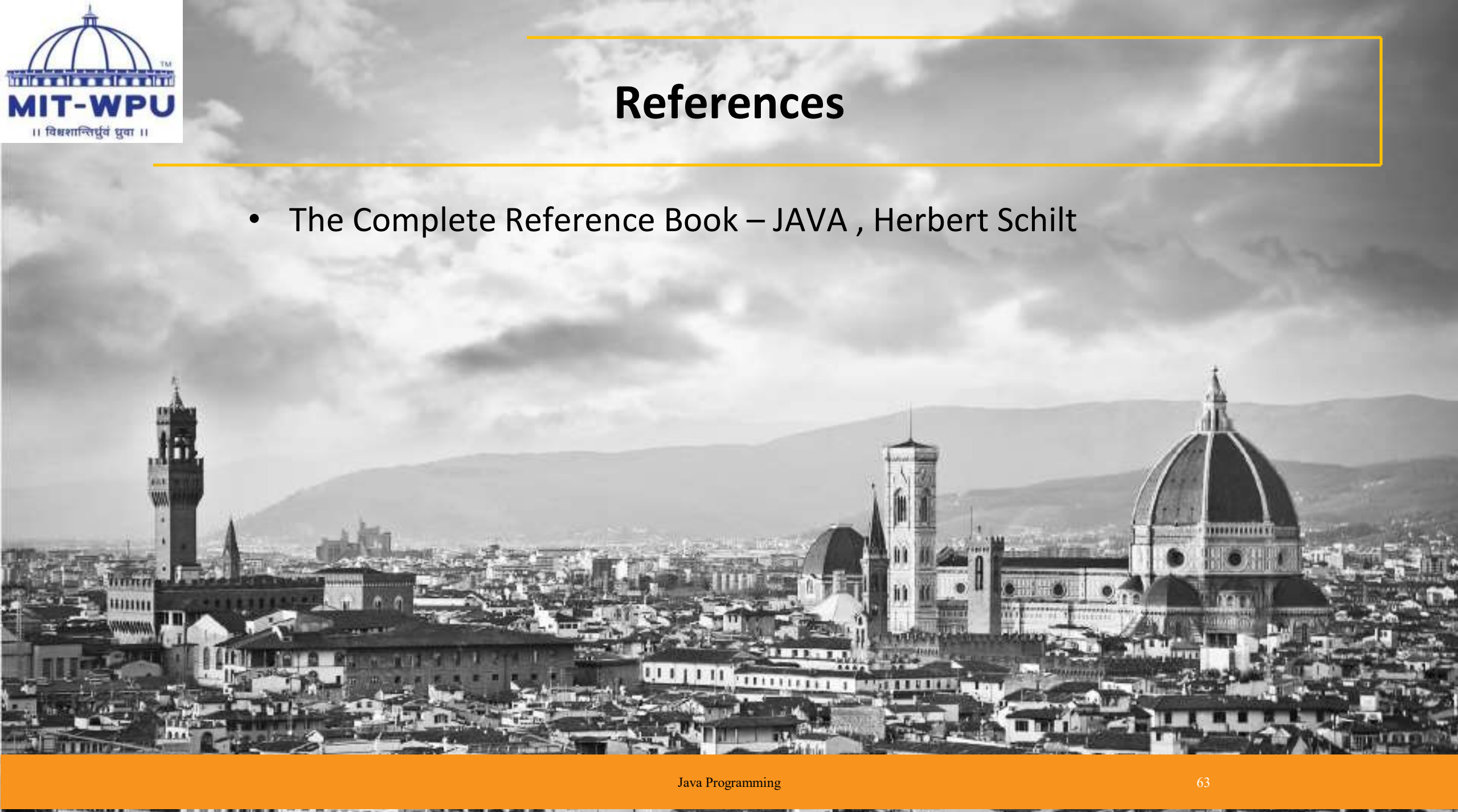
8. NoSuchMethodException: This checked exception is thrown when the undefined method is used in a program.

Key learnings

Keyword	Description
Try	We specify the block of code that might give rise to the exception in a special block with a "Try" keyword.
Catch	When the exception is raised it needs to be caught by the program. This is done using a "catch" keyword. So a catch block follows the try block that raises an exception. The keyword catch should always be used with a try.
Finally	Sometimes we have an important code in our program that needs to be executed irrespective of whether or not the exception is thrown. This code is placed in a special block starting with the "Finally" keyword. The Finally block follows the Try-catch block.
Throw	The keyword "throw" is used to throw the exception explicitly.
Throws	The keyword "Throws" does not throw an exception but is used to declare exceptions. This keyword is used to indicate that an exception might occur in the program or method.

References

- The Complete Reference Book – JAVA , Herbert Schilt




Thank You!!