VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CSI3023- Advanced Server Side Programming

Course Objectives and Outcomes

CSI3023	Advanced Server Side Programming	L	T	P	J	C
		2	0	2	0	3
Pre-requisite	**NIL**	**Syllabus Version**				
		1.0				

Course Objectives:

1. To understand different types of server-side programming and technologies like Servlets, JSP, ASP, EJB, JSF, PHP, Node.
2. Understand the various server-side Spring Frameworks, REST, SOAP, ORM, Security.

Course Outcome:

After successfully completing the course the student should be able to

1. Understand advanced server-side programming concepts and use technologies like Servlets, JSP, JSF and ASP
2. Adopt conveniently, ORM technique to bridge object and relational models of data.
3. Develop, real world API and Services using SOAP and REST.
4. Create application using Node.js and JMS API that provides the facility to create, send and read messages.
5. Efficiently create fast, secure, and responsive web applications using Spring Framework.

Syllabus

Module:1	**Servlets, JSP, JSF and ASP**	**6 hours**
JSP, JSTL, Spring Tag Libraries, Spring Controllers , Template & Layout, Spring Form Validations(Standard and Custom),jQuery, CSS3, Web Descriptor Language, AJAX, Web Socker Support, Java server Faces, JSF flows, UI Model-Framework – JSP, JSTL, Tiles/Thymeleaf, Spring MVC on Spring Boot, Hibernate Validator		
Module:2	**REST**	**3 hours**
Webservices, Types of Webservices, REST, JAX-RS, Rest Frameworks, Rest Methods and APIs, REST Clients.		
Module:3	**SOAP**	**3 hours**
SOAP, JAX-WS, WSDL, SOAP Registries, SOAP Frameworks, SOAP Clients, Develop SOAP and REST API and Services. Framework – Spring MVC, Web-Services, Spring Security		
Module:4	**ORM**	**5 hours**
Object Relation Mapping, JPA, Hibernate, Entity – Annotations, Association and In heritance mapping, Hibernate Session and Transaction, Caching, Native Query, HQL, Batch Processing and Intercepting Filter, Criteria Builder, Projections API, Named & Native Query. Framework – Spring Data JPA, Hibernate and JPA,MySQL/any rdbms Database		

Syllabus

Module:5	**JMS, Node JS**	**4 hours**
JMS, Queues and Topics, Creating Queues and Topics, Sending and Receiving messages using Queues and Topics. Introduction to Node JS, Benefits and Features, NPM in Node JS, Event Handling. Framework – ActiveMQ or RabbitMQ, Spring JMS integration, NodeJS, NPM		
Module:6	**Spring Framework**	**4 hours**
Developing a Batch Application that gets executed in the background process, and gets triggered at a specific regular intervals, Task/Tasklet, Steps, Sharing Batch Context Information between Steps		
Module:7	**Exception Handling**	**3 hours**
Exception Handling, Transaction Commit Intervals, Chunk Processing, File/DB/JMS based Reader and Writers. Framework – Spring Boot, Spring Batch, Spring Data JPA, JMS and MySQL		
Module:8	**Recent Trends**	**2 hours**
	Total Lecture hours:	**30 hours**

Assessment Rubrics

- Digital Assignment -1 (written)
- Quiz-1
- Digital Assignment-2 (Coding)
- CAT-1
- CAT-2
- FAT
- Slow Learners - Assessment

Module –7
Exception Handling

Exception Handling

- **Exception Handling** in Java is one of the effective means to handle runtime errors so that the regular flow of the application can be preserved.
- Java Exception Handling is a mechanism to handle runtime errors such as

  ClassNotFoundException,

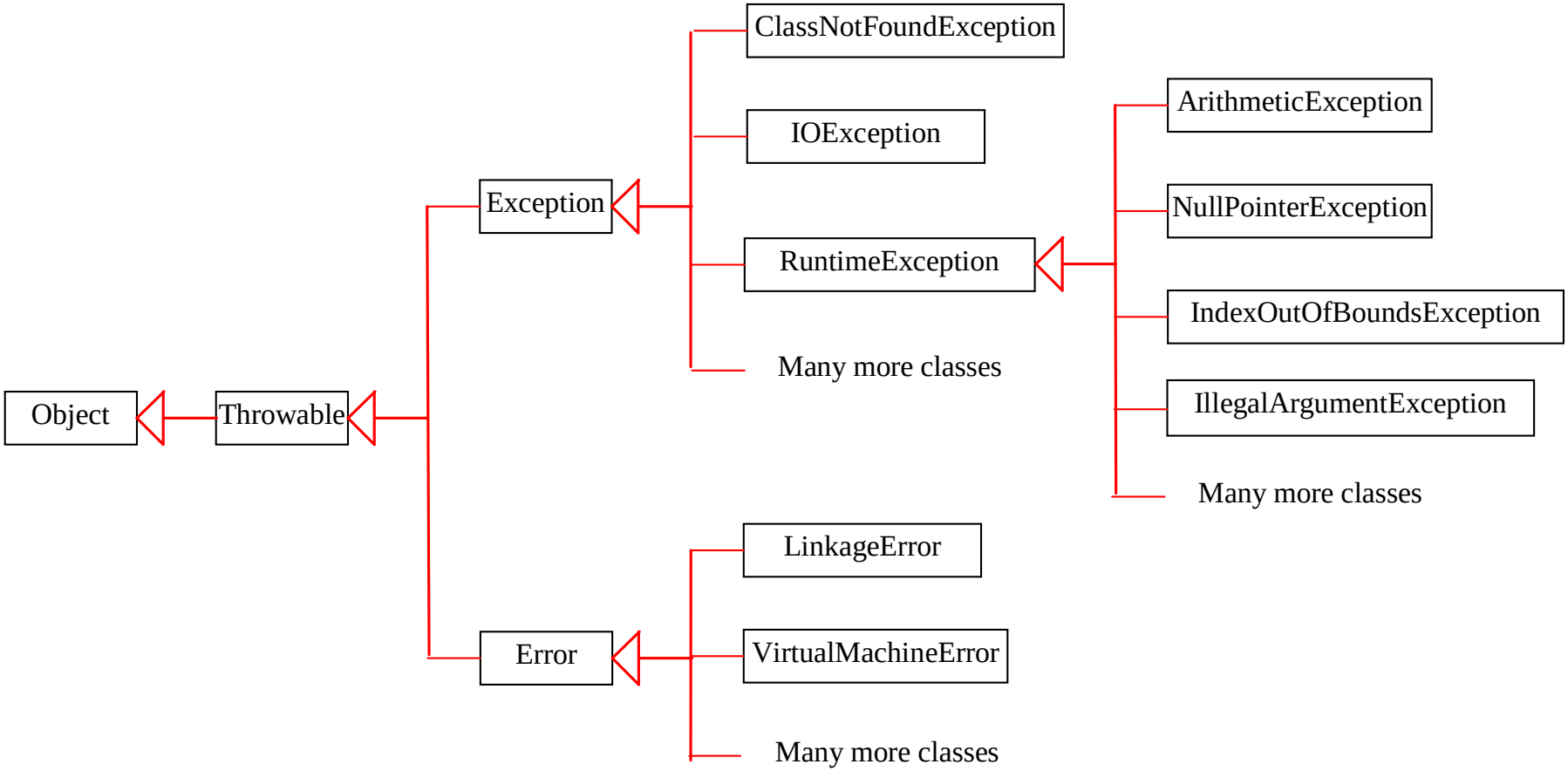
  IOException,

  SQLException, etc.

Exceptions and Errors

- When a problem encounters and unexpected termination or fault, it is called an exception
  - When we try and divide by 0 we terminate abnormally.
- Exception handling gives us another opportunity to recover from the abnormality.
- Sometimes we might encounter situations from which we cannot recover like OutOfMemory. These are considered as errors.

Exception Types



Checked and Unchecked Exception

- Exceptions which are checked for during compile time are called checked exceptions.

  Example: SQLException or any userdefined exception extending the Exception class
- Exceptions which are not checked for during compile time are called unchecked exception.

  Example: NullPointerException or any class extending the RuntimeException class.
- All the checked exceptions must be handled in the program.
- The exceptions raised, if not handled will be handled by the Java Virtual Machine. The Virtual machine will print the stack trace of the exception indicating the stack of exception and the line where it was caused.

Building Blocks
try…catch

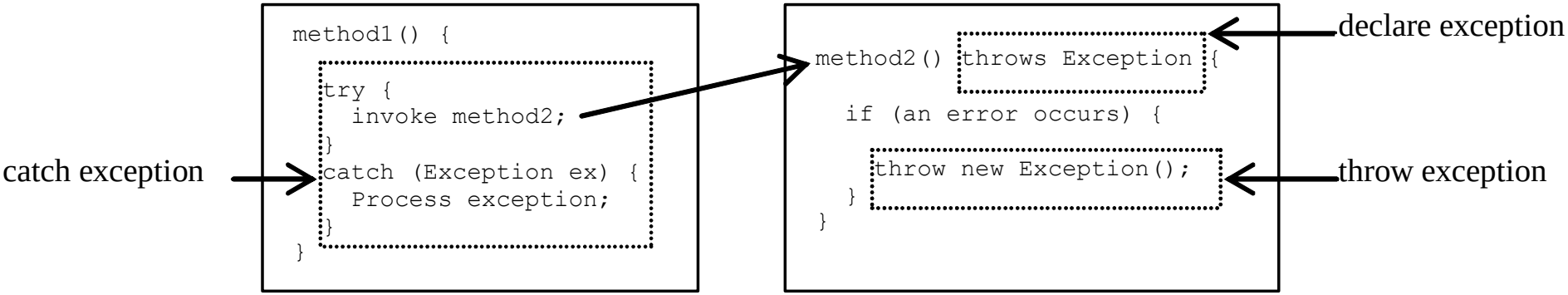
- try block: to define a block of code to be tested for errors while it is being executed.
- catch block:to define a block of code to be executed, if an error occurs in the try block.

```
try {
  //  Block of code to try
}
catch(Exception e) {
  //  Block of code to handle errors
}
```

Declaring, Throwing, and Catching Exceptions

throw and throws keyword:

throw keyword **throws the exception explicitly from a method** or a block of code,
whereas the throws keyword is used in the **signature of the method.**



Demo

```
public class Demo {
  public static void main(String[ ] args) {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]); // error!
  }
}
```

Output:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10
out of bounds for length 3
at javaexception.JavaException.main(JavaException.java:19)
```

A stack trace shows the call stack (sets of active stack frames) and provides information on the methods that your code called. Usually, a stack trace is shown when an Exception is not handled correctly in code.

Demo

```
public class Temp {
        public int x = 10;
                public static void main(String[] args) {
                Temp t = initT();
                int i = t.x;
                }
        private static Temp initT() {
                return null;
        }
}
```

Output:

```
Exception in thread "main" java.lang.NullPointerException
        at Temp.main(Temp.java:9)
```

Demo: try…catch

```
public class Demo {
  public static void main(String[ ] args) {
    try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
    }
   catch (Exception e) {
           System.out.println("Something went wrong.");
    }
  }
}
```

Output:
Something went wrong

finally block

- The `finally` block in Java is a block of code that is used to **execute important code** such as closing the connection, closing the file, or any other cleanup code that needs to be executed regardless of whether an exception occurs or not.

There are three possible cases where the finally block can be used

- **Case 1:** When an exception does not occur, In this case, the program runs fine without throwing any exception and the finally block is executed after the try block.
- **Case 2:** When an exception occurs and is handled by the catch block In this case, the program throws an exception and the catch block handles it. The finally block is executed after the catch block.
- **Case 3:** When an exception occurs and is not handled by the catch block In this case, the program throws an exception but the catch block cannot handle it. The finally block is executed after the try block and then the program terminates abnormally.

Demo : finally

```
public class Demo {
  public static void main(String[] args) {
    try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]);
    } catch (Exception e) {
            System.out.println("Something went wrong.");
    } finally {
            System.out.println("The 'try catch' is finished.");
    }
  }
}
```

Output:
Something went wrong
The 'try catch' is finished

Demo: Arithmetic Exception

```
class demo1
{
    public static void main(String args[])
    {

        int x = 10, y = 0,z;
        try
        {
            z = x / y;
            System.out.println("THE VALUE OF Z is" +z);
        }
        catch (ArithmeticException e)
        {
            String msg = e.getMessage();
            System.out.println("An Error" +msg);
        }
        System.out.println("END");
    }
}
```

Example: Custom Exception

```
public class Demo {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years
      old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
  }

  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

Using Throws Clause

If you don't want the exception to be handled in the same function you can use the throws class to handle the exception in the calling function.

```
public class myexception{
    public static void main(String args[]){
        try{
                checkEx();
        } catch(FileNotFoundException ex){
        }
    }
    public void checkEx() throws FileNotFoundException{
        File f = new File("myfile");
        FileInputStream fis = new FileInputStream(f);
        //continue processing here.
}
}
```

Catching Multiple Exception

```
try {
  statements;// Statements that may throw exceptions
}
catch (Exception1 exVar1) {
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}
...
catch (ExceptionN exVar3) {
  handler for exceptionN;
}
```

Catching Multiple Exception

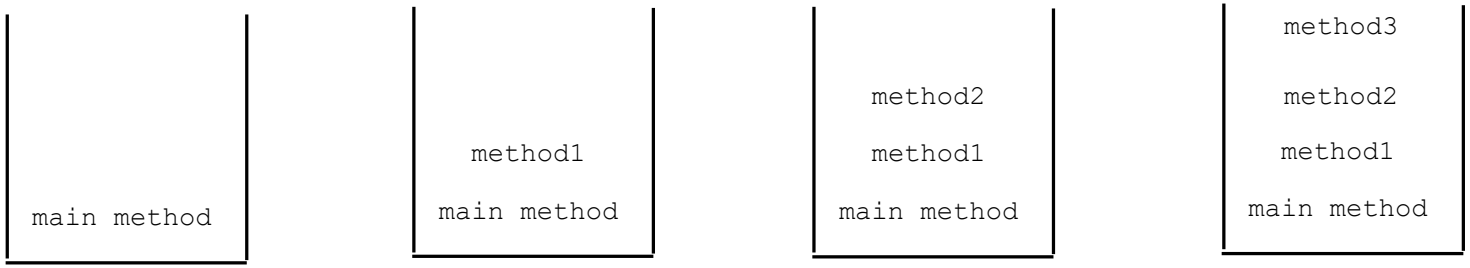
```
main method {
  ...
  try {
    ...
    invoke method1;
    statement1;
  }
  catch (Exception1 ex1) {
    Process ex1;
  }
  statement2;
}
```

```
method1 {
  ...
  try {
    ...
    invoke method2;
    statement3;
  }
  catch (Exception2 ex2) {
    Process ex2;
  }
  statement4;
}
```

```
method2 {
  ...
  try {
    ...
    invoke method3;
    statement5;
  }
  catch (Exception3 ex3) {
    Process ex3;
  }
  statement6;
}
```

An exception is thrown in method3

Call Stack



Catching Multiple exceptions

```
public class myexception{
    public static void main(String args[]){
        try{
        File f = new File("myfile");
        FileInputStream fis = new FileInputStream(f);
        }
        catch(FileNotFoundException ex){
                File f  = new File("Available File");
                FileInputStream fis = new FileInputStream(f);
        }
       catch(IOException ex){
                //do something here
        }
finally{
// the finally block
}
//continue processing here.
    }
}
```

Throwing Exceptions Example

```
/** Set a new radius */
 public void setRadius(double newRadius)
     throws IllegalArgumentException {
   if (newRadius >= 0)
     radius =  newRadius;
   else
     throw new IllegalArgumentException(
       "Radius cannot be negative");
 }
```

Custom or User defined Exception

A Class that represents use-defined exception

```
class MyException extends Exception {
    public MyException(String s)
    {
        // Call constructor of parent Exception
        super(s);
    }
}
```

Custom or User defined Exception

// A Class that uses above MyException

```
public class Demo{
   // Driver Program
   public static void main(String args[])
   {
      try {
         // Throw an object of user defined exception
         throw new MyException(“ My Exception");
      }
      catch (MyException ex) {
         System.out.println("Caught");

         // Print the message from MyException object
         System.out.println(ex.getMessage());
      }
   }
}
```

Custom or User defined Exception

```
/ class representing custom exception
class InvalidAgeException  extends Exception
{
   public InvalidAgeException (String str)
   {
      // calling the constructor of parent Exception
      super(str);
   }
}
```

Custom or User defined Exception

```
// class that uses custom exception InvalidAgeException
public class TestCustomException1
{
    // method to check the age
   static void validate (int age) throws InvalidAgeException{
     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");
      }
    }
```

Custom or User defined Exception

```
// main method
   public static void main(String args[])
   {
      try
      {
         // calling the method
         validate(13);
      }
      catch (InvalidAgeException 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...");
   }
}
```

Quiz

Question -1

What will be the output of the following code?

```
Public class foo{
        public static void main(string args){
        try{
        return;
} finally{
        System.out.println("Finally");
}
}
}
```

A) Compilation fails
B) Finally
C) Return without printing anything
D) none

Question -2

What will be the output of the following code?

```
public class foo{
        public static void main(string args){
        try{
        int x=0;
        int y=5/x;
}catch(Exception e){
        System.out.println("Exception");
} catch(ArithmeticException e){
         System.out.println("Exception");
}
         System.out.println("finished");
}
}
```

A) Compilation fails
B) Exception
C) Arithmeticexception
D) finished

Thank you