Implementing Exception Handling

- When an unexpected error occurs, Java creates an exception type object.
- Once created, Java sends the exception object to the program by throwing the exception.
- The exception object contains information about the type of error and the state of the program when the exception occurred.
- You need to handle the exception by using an exception handler and processing the exception.

Implementing Exception Handling (Contd.)

- You can implement exception handling in a program by using the following keywords:
  - `try`
  - `catch`
  - `throw`
  - `throws`
  - `finally`
  - `try`-with-resources

Implementing Exception Handling (Contd.)

- A `try` block encloses the statements that might raise an exception and defines one or more exception handlers associated with it.
- If an exception is raised within the `try` block, the appropriate exception handler that is associated with the `try` block processes the exception.
- In Java, the `catch` block is used as an exception handler.
- A `try` block must have at least one `catch` block that follows the `try` block, immediately.
- The `catch` block specifies the exception type that you need to catch.

Implementing Exception Handling (Contd.)

- You can declare the `try` and the `catch` block by using the following syntax:

```
try
{
        // Statements that can cause an exception.
}

catch(exceptionname obj)
{
        // Error handling code.
}
```

Implementing Exception Handling (Contd.)

- In case, you need to accept two integers from a user, perform their addition, and display the result, you can use the following code:

```
import java.util.Scanner;
public class Addition {
     public static void main(String[] args) {
        int num1, num2, result;
        Scanner obj1 = new Scanner(System.in);
        System.out.println("Enter the 1st number");
        num1 = obj1.nextInt();
        System.out.println("Enter the 2nd number");
        num2 = obj1.nextInt();
        result = num1+num2;
        System.out.println("The result is
        "+result);
        }
     }
```

Implementing Exception Handling (Contd.)

- To implement exception handling by using the try-catch block, you can use the code given in the embedded document:

Implementing Exception Handling (Contd.)

- A `try` block can have multiple `catch` blocks.
- You can declare multiple `catch` blocks with a single `try` statement by using the following code snippet:

```
try
{
// statements
}
catch(exceptionname1 obj1)
{
//statements to handle the exception
}

catch(<exceptionname2 obj2)
{
//statements to handle the exception
}
```

Implementing Exception Handling (Contd.)

```
catch(exceptionnameN objN)
{
//statements to handle the exception
}
```

- While working with multiple `catch` statements, it is important to follow the exception hierarchy, such that the subclasses must appear prior to the superclasses.
- If the exception hierarchy is not followed, a compile-time error is generated.
- In order to generate a compile-time error if the exception hierarchy is not followed, you can use the code given in the embedded document:

Microsoft Word Document

Implementing Exception Handling (Contd.)

- You can throw an exception explicitly by using the `throw` keyword.
- The `throw` keyword causes the termination of the normal flow of control of the Java code and stops the execution of the subsequent statements.
- You can throw an exception by using the following syntax:

```
throw ThrowableObj
```

- Consider the following code that demonstrates the implementation of the `throw` statement:

```
public class ThrowDemo
{
    void display()
    {
        throw new RuntimeException();
    }
```

Implementing Exception Handling (Contd.)

```
public static void main(String[] args)
    {
        ThrowDemo obj1 = new ThrowDemo();
        try
        {
            obj1.display();
        } catch (RuntimeException e)
        {
            System.out.println("Runtime Exception
raised");
        }
    }
}
```

Implementing Exception Handling (Contd.)

- The `throw` keyword can also be used inside a `catch` block to rethrow an exception.
- You can use the following code snippet to rethrow an exception:

```
catch(Exception e)
{
        System.out.println("Exception Raised");
throw e;
}
```

- To catch the `RuntimeException` exception and rethrow the exception to the outer handler, you can use the code given in the embedded document:

Implementing Exception Handling (Contd.)

- The `throws` keyword is used by a method to specify the types of exceptions that the method can throw.
- The `throws` keyword lists the checked exceptions that a method can throw.
- You can use the following syntax to declare a method that specifies a `throws` keyword:

```
<access_specifier> <modifier> <return_type>
<method_name> (<arg_list>) throws <exception_list>
```

- Consider the following code that demonstrates the implementation of the `throws` statement:

```
public class ThrowsDemo
{
      void display() throws Exception
         {
         throw new Exception();
         }
```

Implementing Exception Handling (Contd.)

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

        ThrowsDemo obj1 = new ThrowsDemo();
        try
           {
           obj1.display();
           }
        catch (Exception e)
           {
           System.out.println("Runtime Exception
raised");
           }
     }
}
```

Implementing Exception Handling (Contd.)

- During the execution of a java program, when an exception is raised, the rest of the statements in the `try` block are ignored.
- Sometimes, it is necessary to execute certain statements, irrespective of whether an exception is raised.
- The `finally` block is used to execute these required statements. The statements specified in the `finally` block are executed after the control has left the try-catch block.

Implementing Exception Handling (Contd.)

- You can use the following syntax to declare the `try` and `finally` block:

```
try
{
    // Block of code
}
finally
{
// Block of code that is always executed
irrespective of an exception being raised.
}
```

Implementing Exception Handling (Contd.)

- If there is a `catch` block associated with the `try` block, the `finally` block is written after the `catch` block.
- The following code snippet shows the syntax to declare the `try`, `catch`, and `finally` blocks:

```
try
{
        // Block of code.
}
catch(execptionname1 obj1)
{
        System.out.println("Exception1 has been
raised");
}
catch(exceptionname2 obj2)
{
```

Implementing Exception Handling (Contd.)

```
System.out.println("Exception2 has been raised");
}
finally
{
// Block of code that is always executed
irrespective of an
 exception being raised or not.
}
```

- The `finally` block executes irrespective of whether or not an exception is raised. If an exception is raised, the `finally` block executes even if none of the `catch` blocks match the exception.
- The `try`-with-resources statement is similar to the `try` block.

Implementing Exception Handling (Contd.)

- However, it is essentially used to declare and automatically close the objects, such as the file streams and database connections after the execution of the `try` block finishes. Such objects are known as resources.
- In order to be handled by the `try`-with-resources statement, the resource must implement the `java.lang.AutoCloseable` interface.
- The `try`-with-resources block ensures that one or more system resources are released when they are no longer required.

Implementing Exception Handling (Contd.)

- You can use the following syntax to declare the `try`-with-resources statement:

```
try( [resource-declaration 1];
[resource-declaration n];
)

{
//code to be executed
}
//after the try block, the resource is closed
```

- The following code snippet shows how to implement the `try`-with-resource statement:

```
try (BufferedReader br = new BufferedReader(new
FileReader("<file_path>")))
{
        return br.readLine();
}
```

User-defined Exceptions

- In addition to the built-in exceptions, you can create customized exceptions, as per the application requirements.
- To create a user-defined exception, you need to perform the following steps:
  - Create an exception class.
  - Implement user-defined exception.
- If you want to create a new user-defined exception, the class should extend the `Throwable` class or its subclasses.

User-defined Exceptions (Contd.)

- Consider the following code snippet:

```
public class AgeException extends RuntimeException
{
     public AgeException()
       {
       System.out.println("Invalid value for age");
       }
AgeException(String msg)
{
                        super(msg);
       }
}
```

User-defined Exceptions (Contd.)

- Consider the following code to demonstrate the implementation of a user-defined exception:

```
import java.util.*;
public class ValidateAge
{
  public static void main(String[] args)
{

        int age;
        Scanner obj1 = new Scanner(System.in);
        System.out.println("Enter the age: ");
        age = obj1.nextInt();
         if (age <= 0){
            try
{
```

User-defined Exceptions (Contd.)

```
        throw new AgeException();
                                }
                    catch (AgeException e)
                                {

                       System.out.println("Exception raised");

                                }
                }
                     else
        {
                                System.out.println("Age entered
        is " + age);
                                }

                }

        }
```

Just a minute

- Which one of the following keywords lists the checked exceptions that a method can throw?
  - `throws`
  - `throw`
  - `catch`
  - `finally`

Just a minute (Contd.)

- Solution:
  - `throws`