Exception Handling

Errors:

Errors are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

An exception:

An exception is a problem that arises during the execution of a program. When an **Exception** occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Types of Exceptions.

1. **Compile time/ Checked exception.**
   A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the programmer should handle these exceptions.

2. **Runtime/ unchecked exception.**

An unchecked exception is an exception that occurs at the time of execution. These are also called as **Runtime Exceptions**. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

Exception Handling

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

In exception handling we need to use the following terminologies.

1. **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.

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{}
```

2. **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.

```
try{
// protected code
}catch(ExceptionType1  e1){
}
```

The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception occurred is handled by catch block associated with it. Every try block should be immediately followed either by a catch block or finally block.

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked.

Multiple Catch Blocks

A try block can be followed by multiple catch blocks.

```
try{
// protected code
}catch(ExceptionType1  e1){
} catch(ExceptionType2   e2){
} catch(ExceptionType3  e3){
}
```

If an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the type of the exception thrown matches ExceptionType1, it gets caught there. If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls through all catches, in which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

3. **finally block**

The finally block always get executed despite the exception occurred or not.

The finally block is useful to close the database connection or close a file, etc.

Syntax:

```
finally{

//statements to release the resources

}
```

4. **throws keyword:**

If a method does not handle a checked exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

Ex:

```
public void insertRecord() throws SQLException{

//method body contains code that raises checked exceptions SQLException.

}
```

5. **throw keyword:**

You can throw an exception, either a newly instantiated one or an exception that you just caught, by using the **throw** keyword.

Try to understand the difference between throws and throw keywords, *throws* is used to postpone/forward the handling of a checked exception and *throw* is used to invoke an exception explicitly.

Custom / User defined 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.

Simple example of java custom exception.

```
//creating a custom exception class
class InvalidUsernameException extends Exception{
 InvalidUsernameException(String s){
  super(s);
 }
}
class CustomExceptionDemo{
   static void validateUsername(String username)throws InvalidUsernameException {
     if(username. length()<5)
      throw new InvalidUsernameException("not a valid username, must be greater than
4 character"); // creating and throwing an object of our custom exception class
     else
      System.out.println("Valid,  welcome");
   }

   public static void main(String args[]){
      try{
      validateUsername(“abc”);
      }catch(Exception m){System.out.println("Exception occured: "+m);}

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