Managing Errors and Exceptions

1. **Write the different types of errors that occurred in the Java program. What are the causes of such type of errors?**

Ans:

Errors may broadly be classified into two categories:

1. Compile-time errors
2. Run-time errors

Compile-Time Errors:

All syntax errors will be detected and displayed by the java compiler and therefore these errors are known as compile-time errors. Whenever the compiler displays an error, it will not create the `.class` file. The most common compile-time errors are:

- Missing semicolons
- Missing (or mismatch of) brackets in classes and methods
- Missing double quotes in strings
- Use of undeclared variables
- Incompatible types in assignments / initializations
- Bad references to objects
- Use of = in place of == operator
- And so on

Run-Time Errors:

Sometimes, a program may compile successfully creating the `.class` file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow. The most common run-time errors are:

- Dividing an integer by zero
- Accessing an element that is out of the bounds of an array
- Trying to store a value into an array of an incompatible class or type
- Trying to cast an instance of a class to one of its subclasses
- Trying to illegally change the state of a thread
- Attempting to use a negative size for an array
- Using a null object reference as a legitimate object reference to access a method or a variable
- Converting an invalid string to a number
- Accessing a character that is out of bounds of a string and many more.

2. What is exception? What are the differences between Errors and Exceptions in Java?
Ans:

Exception:
An exception is a condition that is caused by a run-time error in the program. When the java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it (i.e.; informs us that an error has occurred).

Differences between Errors and Exceptions in Java:

- Errors may be compile time or run-time errors but an exception is a run-time error.
- An error is an irrecoverable condition occurring at runtime, such as an OutOfMemory error. These are JVM errors and you cannot repair them at runtime. Though errors can be caught in the catch block, the execution of the application will come to a halt and is not recoverable.

  While exceptions are conditions that occur because of bad input etc. e.g. `FileNotFoundException` will be thrown if the specified file does not exist. In most of the cases it is possible to recover from an exception (probably by giving the user a feedback for entering proper values etc.)

3. Write about six common Java exceptions with the cause of occurrence.
Ans:

Common Java Exceptions:

Exception Type	Cause of Exception
`ArithmeticException`	Caused by math errors such as division by zero.
`ArrayIndexOutOfBoundsException`	Caused by bad array indexes.
`FileNotFoundException`	Caused by an attempt to access a nonexistent file.
`IOException`	Caused by general I/O failures, such as inability to read from a file.
`NullPointerException`	Caused by referencing a null object.
`OutOfMemoryException`	Caused when there's not enough memory to allocate a new object.
`StackOverflowException`	Caused when the system runs out of stack space.

4. Explain the exception handling mechanism.

Or

Write the general form of the exception-handling block in Java.

Ans:

If the exception object is not caught and handled properly, the interpreter will display an error message. If we want the program to continue with the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as exception handling.

The basic concepts of exception handling are throwing an exception and catching it. This is illustrated in the figure below:

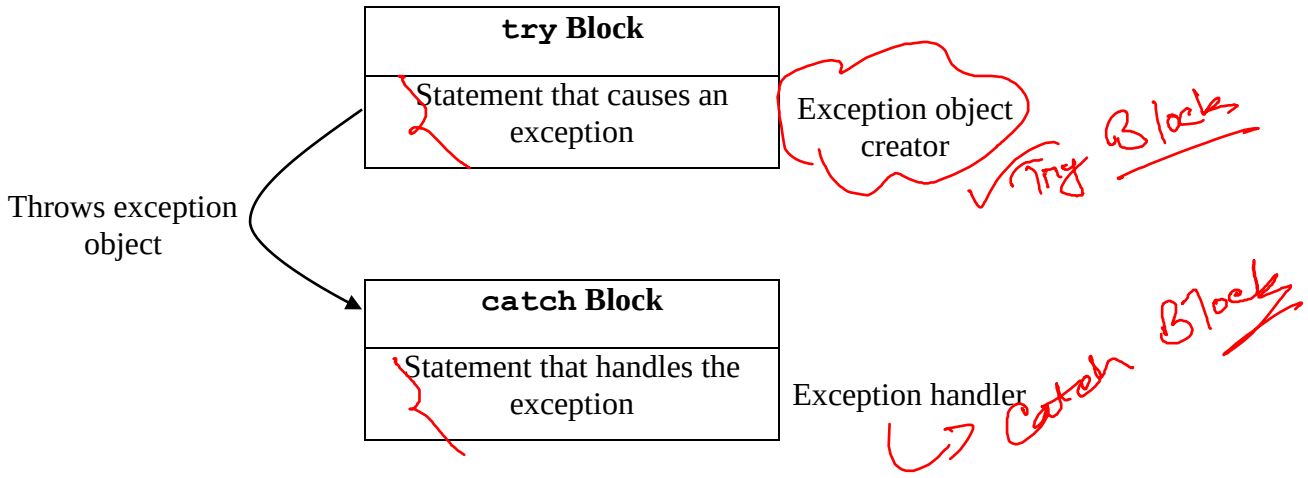

Fig: Exception handling mechanism.

```
try
{
      statement;   // generates an exception
}

catch(Exception-type e)
{
      statement; //  processes the exception
}
```

The `try` block can have one or more statements that could generate an exception. If any statement generates an exception, the remaining statements in the block are skipped and execution jumps to the `catch` block that is placed next to the `try` block.

5. How can you handle an exception that is not caught by any of the previous catch statements? Give example.

Or

What is the finally block? When and how it is used? Give example.

Or

Explain the use of the finally block with an example.

In Java `finally` statement can be used to handle an exception that is not caught by any of the previous `catch` statements; `finally` block can be used to handle any exception generated within a `try` block. It may be added immediately after the `try` block or after the last `catch` block shown as follows:

```
try{
      . . . .
      . . . .
}


finally{
      . . . .
      . . . .
}
```

```
try{
      . . . .
      . . . .
}

catch(...){
      . . . .
      . . . .
}

catch(...){
      . . . .
      . . . .
}

finally{
      . . . .
      . . . .
 }
```

When a `finally` block is defined, this is guaranteed to execute, regardless of whether or not an exception is thrown. As a result, we can use it to perform certain housekeeping operations such as closing files and releasing system resources.

Example:

```
public class FinallyExample {
      public static void main(String args[]){
            int a[]={5,10};
            int b=5;

            try{
                  int x=a[2]/b-a[1];
            }

            catch(ArithmeticException e){
                  System.out.println("Division by Zero");
            }

            catch(ArrayIndexOutOfBoundsException e){
                  System.out.println("Array Index Error");
            }

            catch(ArrayStoreException e){
                  System.out.println("Wrong data type");
            }

            finally{
                  int y=a[1]/a[0];
                  System.out.println("y= "+y);
            }
      }
}
```

6. With an example illustrating the usage of throw and throws in a Java program.
Ans:

throw:

`Throw` keyword is used to throw the exception manually. It is mainly used when the program fails to satisfy the given condition and it wants to warn the application. The exception thrown should be a subclass of `Throwable`.

```
      public void saveUserAge(int age) {
            if(age<18){
                  throw new ArithmeticException();
            }
            else{
                  System.out.println("Correct are is entered");
            }
       }
```

throws:

If the function is not capable of handling the exception then it can ask the calling method to handle it by simply putting the `throws` clause at the function declaration.

```
class ThrowDemo1{
     static void play()  throws Exception{
          System.out.println("before");
          throw new IllegalAccessException("demo");
          //System.out.println("after");
     }

     public static void main(String args[]){
           try{
              play();
            }

            catch(Exception e){
              System.out.println("caught  me: " +e);
            }
     }
}
```

7. Write the differences between throw and throws in Java.

Ans:

Differences between throw and throws in Java:

throw	**throws**
`throw` is used to explicitly throw an exception.	`throws` is used to declare an exception.
A checked exception cannot be propagated without `throw`.	A checked exception can be propagated with `throws`.
`throw` is followed by an instance.	`throws` is followed by class.
`throw` is used within the method.	`throws` is used with the method signature.
You cannot `throw` multiple exceptions.	You can declare multiple exceptions.

8.

```
public class Error {
	public static void main(String args[]){
		int a[]={7,9,45,63,10};
		int b=7;

		for(int i=0;i<7;i++){
			int x=a[i]/(b-7);
			System.out.print(x);
		}
	}
}
```

What types of exceptions may occur in the above code and how can you handle those exceptions, write your solution code.

Ans:

Two types of exceptions may occur in the above code. These are `ArithmeticException` and `ArrayIndexOutOfBoundsException`.

Java uses a keyword `try` to preface a block of code that is likely to cause an error condition and "throw" an exception. A catch block is defined by the keyword `catch` "catches" the exception "thrown" by the try block and handles it appropriately.

Solution Code:

```
public class Error {
	public static void main(String args[]){
		int a[]={7,9,45,63,10};
		int b=7;

		try{
			for(int i=0;i<7;i++){
				int x=a[i]/(b-7);
				System.out.print(x);
			}
		}

		catch(ArithmeticException e){
			System.out.println("Division by Zero");
		}

		catch(ArrayIndexOutOfBoundsException e){
			System.out.println("Array Index Error");
		}
	}
}
```

9. Define an exception called "NoMatchException" that is thrown when a string is not equal to "BANGLADESH". Write a Java program that uses the exception.

Or

Describe with an example how we can throw our own exception.

Ans:

We can throw our own exception by using the keyword `throw` as follows:

```
throw new Throwable_subclass;
```

The below example illustrate this.

```
import java.lang.Exception;
import java.util.Scanner;

class NoMatchException extends Exception{
    NoMatchException(String message){
        super(message);
    }
}

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

        Scanner scan = new Scanner(System.in);

        String s1="BANGLADESH";
        String s2=scan.next();

        try{
            if(s2.equals(s1))
                System.out.println("Match");
            else
                throw new NoMatchException("Not Match");
        }

        catch(NoMatchException e){
            System.out.println(e.getMessage());
        }

        finally{
            System.out.println("Finally Block");
        }
    }
}
```

10. Define an exception called "ValueExceeds" that is thrown when the integer input taken from the user is greater than 100. Write a program that uses this exception.

Ans:

```
import java.lang.Exception;
import java.util.Scanner;

class ValueExceeds extends Exception{
     ValueExceeds(String message){
          super(message);
     }
}

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

          Scanner scan = new Scanner(System.in);
          int i=scan.nextInt();

          try{
             if(i>100)
               throw new ValueExceeds("Input Greater than 100");
          }

          catch(ValueExceeds e){
               System.out.println(e.getMessage());
          }

          finally{
               System.out.println("Finally Block");
          }
     }
}
```