CHAPTER 11

1. **1. Division by Zero**

* **Description**: Attempting to divide a number by zero.
* **How It's Handled**:
  + In most languages, division by zero results in a runtime exception (e.g., ArithmeticException ).
  + To handle this, the program can:
    - **Use try-catch blocks**: Enclose the division operation in a try block, catching the exception in a catch block to handle the error gracefully.
    - **Check before division**: A condition can be checked before the operation, ensuring the denominator is not zero.

2. **2. Array Index Out of Bounds**

* **Description**: Accessing an element in an array using an invalid index (e.g., an index greater than or equal to the array's length).
* **How It's Handled**:
  + This typically raises an IndexOutOfBoundsException .
  + Programs should:
    - **Catch the exception**: Wrap the array access code in a try-catch block and handle the exception in the catch block.
    - **Check index validity**: Ensure the index is within the valid range before attempting to access the array element.

3. **3. Null Pointer Dereference (or NullReferenceException)**

* **Description**: Accessing a method or property on a null object reference.
* **How It's Handled**:
  + This raises a NullPointerException.
  + Programs can:
    - **Check for null**: Before accessing the object, check whether it is or null .
    - **Use exception handling**: If a null reference might occur, enclose the code in a try-catch block.

4. **4. File Not Found**

* **Description**: Trying to open or access a file that does not exist.
* **How It's Handled**:
  + This raises a FileNotFoundException .
  + Programs should:
    - **Use try-catch**: Wrap the file access code in a try-catch block to handle the error.
    - **Check file existence**: Before opening a file, check if it exists

5. **Illegal Argument Exception**

* **Description**: Passing an illegal argument to a function or method (e.g., negative numbers when only positive values are allowed).
* **How It's Handled**:
  + This results in an IllegalArgumentException.
  + Programs should:
    - **Validate input**: Before performing operations, ensure arguments meet expected conditions.
    - **Throw custom exceptions**: If an illegal argument is detected, throw a custom exception or return an error message.

**2 Exception handling is an effective means for dealing with constructor failure because it provides a clean, structured way to manage errors without disrupting the flow of the program. Here's why:**

**1. Separation of Concerns:**

* **Without Exception Handling: When a constructor fails (e.g., invalid arguments or internal errors), you often have to return a special error code, set a flag, or perform complex checks to indicate failure. This creates clutter within the constructor logic and makes the code harder to read and maintain.**
* **With Exception Handling: Exception handling allows the constructor to simply throw an exception when it encounters an error, leaving the rest of the program to handle the failure in a more structured way. This separates the creation of the object from error management.**

**2. Clearer Error Propagation:**

* **Without Exception Handling: If a constructor cannot properly initialize an object, it may return a null object or a partially initialized instance. This can lead to subtle bugs where the calling code doesn’t realize that the object is in an inconsistent state. Additionally, handling errors through flags or error codes can lead to confusion because it’s not immediately obvious what went wrong.**
* **With Exception Handling: When an exception is thrown by a constructor, it automatically propagates up the call stack, signaling the failure. The caller can then catch the exception and take appropriate action (such as logging, retrying, or gracefully shutting down). This makes the error more visible and easier to address.**