Exception Handling

Session 6

Contents

- Types of errors.
- Exceptions
- Need of exception handling
- Exception handling in Java

Errors

- There is no perfect world.
  - Errors are inevitable.
- Errors are shipped even with best software.
- Error occurs due to
  - Wrong input.
  - Error on platform or system not configured properly
  - Bugs in program.

  Occurs when application is operational.

Types of errors

- Logical errors
  - Counter not incremented in loop.
  - Wrong expression resulting incorrect input
- Compile time errors
  - Wrong syntax.
  - Variable not declared
  - ; missing
- Run time errors
  - Division by zero
  - Memory allocation fail
  - File not found
  - Array index is out of bound

What should a good software should provide?

- Software should be robust. (Error free)
- If error occurs in application, it should
  - Give understandable error message
  - Save all the data
  - Gracefully shut down.

Exceptions

- Errors that occur during execution of program are known as **run time errors** or **exceptions**.
- For example
  - Divide by zero
  - File not found
  - Array index out of its bound
  - File is corrupt.
  - Invalid type casting is done
  - Data entered is not in correct format.

Exceptions

- Java supports two types of exceptions.
- Checked exceptions
  - These type of errors occur due to problem with resources of system settings.
    - FileNotFoundException, IOException etc.
- Unchecked exceptions
  - These type of error occur due to programmers mistake
  - Also known as runtime exception
    - NullPointerException, ArrayOutOfRangeException etc.

Robust error handling

- Run time error handling can be implemented by a mechanism called as **Exception Handling**.
- Can be handled in following steps.

1. Identify block which may cause an error
2. How to handle
3. Differentiate between errors.
4. Mechanism to check resource leakage.
5. What if not handled?

Identify block of code

- Code is executed assuming that there are no errors.
- If some error is expected, it is enclosed within 'try' block.

This statement may cause an exception, so enclosed within 'try'

```
int[] arr = new int[5];
try{
System.out.println(arr[5]);
}
```

How to handle?

- If error occur in the ‘try’ block, handling of error should be done.
  - Error handling code is enclosed in ‘catch’ block

```
int[] arr = new int[5];
try{
System.out.println(arr[5]);
}
catch(ArrayIndexOutOfBoundsException ex){
System.out.println("Array index is out of range");
}
```

Differentiate between exceptions

- Multiple exceptions are can occur in ‘try’ block.
  - Errors needs to be distinguished.
  - Should be handled by using multiple ‘catch’ blocks

```
int[] arr = new int[5];
try{
System.out.println(arr[5]);
}
catch(ArrayIndexOutOfBoundsException ex){
System.out.println("Array index is out of range");
}
catch(NullPointerException ex){
System.out.println("Something went wrong");
}
```

Catch all errors

- Object of **Exception** class can handle all other exceptions that are not handled by any 'catch' block

```
int[] arr = new int[5];
try{
System.out.println(arr[5]);
}
catch(HeadlessException ex){
System.out.println("Array index is out of range");
}
catch(NullPointerException ex){
System.out.println("Something went wrong");
}
catch(Exception ex){
System.out.println("Problem with code");
```

Mechanism to check resource leakage

- Some resources that are used in ‘try’ needs to be released to avoid resource leakage.
- This can be handled via ‘finally’ block.
- ‘finally’ block executed irrespective of error.

Mechanism to check resource leakage

```
int[] arr = new int[5];
try{
System.out.println(arr[5]);
}
catch(HeadlessException ex){
System.out.println("Array index is out of range");
}
catch(NullPointerException ex){
System.out.println("Something went wrong");
}
catch(Exception ex){
System.out.println("Problem with code");
}
finally{
System.out.println("Finally block executed!!!");
}
```

What if not handled?

- If any of the exception is unable to handled then it is propagated to up in the stack ie. to the calling method

main()

```
try{
method1();
}
catch(...){

}
```

method1()

```
try{
method2();
}
catch(...){

}
```

method2()

```
int ans = 100 / 0;
```

Error Occured


Thank
You!