# Exception handling

## Types of anomalies

These are the different types of anomalies that can happen in a program:

1. Syntax Error
- Errors that arise from wrong syntax
- Also known as compile time error.
- These are severe in nature.
- Program wont be compiled without removing them
- These can be detected by IDE or by Java Compiler.

2. Warning
- These are intimations to structure the code.
- These are not severe.
- These are also detected by IDE.

3. Semantic/Logical Error
- Errors arising due to wrong logic.
- These could not be detected by IDE.
- These can be detected by testing the code by execution or observation
- Their severity depends on business requirements

4. Runtime Error
- Errors arising due to wrong data, failure in connection, setup mistakes
- These are runtime errors
- These have to be handled by programmer

----X-----X-----X-----

## What is an Exception?

- The term exception is shorthand for the phrase "exceptional event."

- It is an event that disrupts the normal flow of the program and causes its abnormal termination.

## Causes of Exception

Exceptions in java can arise from different kind of situations such as:

- Wrong data entered by user,

- Hardware failure,

- Network connection failure,

- Database server down etc.



## Java Exeption Mechanism

All the exceptions that can be thrown are already known in Java and so there is a mechanism in place.

For every exception there is a class.

Whenever an exception occurs, 2 things happen:
1. An exception object is created
2. This object is thrown

for ex: When in code 10/0 happens

1. JRE creates an object of ArithmeticException class
ArithmeticException a = new ArithmeticException();

2. JRE throws this exception: throw a;

Now as a developer our job is to catch the exception and handle it in a rational manner

If the exception is not caught, program will terminate at the point of exception.

Exception Hierarchy
---
All the exception classes are arranged in a hierarchy

Refer figure - 01_excp_hierarchy.png

Throwable is the root class of Exception and Errors

Exception is the root class of Exception Mechanism Hierarchy

Types of Exceptions
---
There are 2 types of exceptions:

1. Runtime Exceptions
    - All the exception classes that extend the RuntimeException class
    - Handling them is optional.
    - AKA Unchecked exceptions (not checked by compiler)

2. Compile Time Exceptions:
    - Handling them is mandatory
    - Compiler checks whether certain conditions are handled before executing them.
    - AKA Checked Exceptions

    - Generally when our program deals with outside resources (that are beyond its control), it assumes that problems will come. So handling them is compulsory



Exception Handling Keywords
==========
Java provides 5 keywords to handle exceptions - try, catch, throw, throws, finally

Java try-catch block
---

```
In the try block
      Enclose the code that we think may produce an exception
In the catch block
      Handle the exception thrown
      We can also use the parent classes of exception thrown
      The exception won't be handled by sibling classes or non related
classes

StackTrace
==========
- The list(Stack is a List) of method calls that have been used in a
particular operation is known as Stacktrace.

- The stack trace contains all the information of the exception
(exception type, description, some time suggestions to fix it, line
number, exact point of error)

- Here we have to identify the class and method that is under our control
(so that we can check the data, handle the exception, validate the data)

Ex:

Scanner s = new Scanner(System.in);
S.o.p("\n Enter number: ");
int number = s.nextInt();

Run code:
 Enter number: 1.2

Logs:
Exception in thread "main" java.util.InputMismatchException
      at java.base/java.util.Scanner.throwFor(Scanner.java:939)
      at java.base/java.util.Scanner.next(Scanner.java:1594)
      at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
      at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
      at
ex01.anomalies.Ex02ExceptionExamples.main(Ex02ExceptionExamples.java:35)

Try catch use cases
-------------------
- If you want to handle all the exceptions in only 1 way, you can specify
the parent class of all exceptions that may come. catch(Exception a){}

- If you want to handle all the exceptions specifically, you can specify
multiple catch blocks.
catch(ArithmeticException a){}
catch(NullPointerException n){}
catch(StringIndexOutOfBoundsException s){}

- If you want to group some catches together, use pipe (|) operator.
catch(ArithmeticException | NullPointerException n){}
catch(StringIndexOutOfBoundsException s){}

static method
```

```
-------------
Within a class, we cannot call a non-static method from within another
static method.
Static methods are used for 2 purposes:
1. When we want to call it directly by the name of the class.
For Ex: Math.pow(), Park.calculateGrassCost()

2. When we want do write a functionality that is not specific to 1
object.
For Ex: Employee.calculateTotalSalaries(), Height.addAllHeights()

calculateTotalSalaries and addAllHeights is not the property of 1 object
but all objects.

Throwing an exception
---------------------
As Java implicitly throws an exception so could we.
To do so we use throw & throws keyword.

Throw
-----
- throw is used from within a method to throw an exception explicitly
- We can throw both - checked and unchecked exceptions

if(age<18) throw new InvalidAgeException("Age less than 18");

Throws
------
throws is used to pass-on/delegate the reponsibility of handling the
exception to calling method

void validateAge() throws InvalidAgeException {
      if(age<18)
            throw new InvalidAgeException("Age less than 18");
}

Note:
- If we throw Unchecked exception, program will compile but when program
is executed, we get exception.

- If we throw Checked exception, program will not compile unless we
either handle it or declare the method with throws clause.

Custom Exception
================
- We can create our own exceptions.
- It can be achieved by deriving classes of Exception.

Steps
-----
There are 3 steps to create a custom exception:

Step 1. Define the exception class
      - Define a simple class: InvalidAgeException
```

- Extend this class from any exception class

Step2. Define atleast 1 constructor

Step3. Specify throw in exception-producing method
Use throws keyword to declare the method which will be producing exception.

Step 4. Handle th0e exception in calling Method

finally block
==========
1. finally ensures that a block of code is executed, whether an exception is thrown or not.

2. It is commonly used to release resources like files, databases, or network connections.

3. finally guarantees that the code within it will run, providing a reliable way to handle cleanup or finalization tasks.

Try with resources
==========
- It makes it easier to work with resources like files, databases, or network connections.

- It automatically takes care of opening and closing these resources, reducing the chances of resource leaks or errors in your code.

- We put the resources in the try block, and Java ensures they are closed properly when we're done, making our code cleaner and safer.

Syntax:

```
try (Scanner s = new Scanner(System.in)){

} catch(){

}
```

Note: Only those classes can be closed with try with resources that implements the AutoCloseable Interface

Optional Class
==========
- Optional class was introduced in Java 8
- It is a public final class and used to deal with NullPointerException
- It can help in writing a neat code without using too many null checks.
- With Optional, we can specify alternate values to return, or alternate code to run
-----X-----X-----X-----X-----
Debugging Code
==========
Debugging the code: Knowing/Removing/Solving errors

1. Know the concepts well
2. Write good code first up
3. Know the code flow
4. Be observant towards our IDE, Red is Syntax Error, Orange/Yellow is warning - remove them
5. Format your code: Ctrl + Shift + F
6. Follow the conventions: Naming Conventions, Indentations, Create Packages
7. Spelling mistakes have no effect if they match, but they must match
8. Console is our friend - Read it well, it gives all the information: Name of page, line no, exception name everything.
9. Use Eclipse Debugging tool