Spring Chitkara University Live [SCUL] 📚⚽

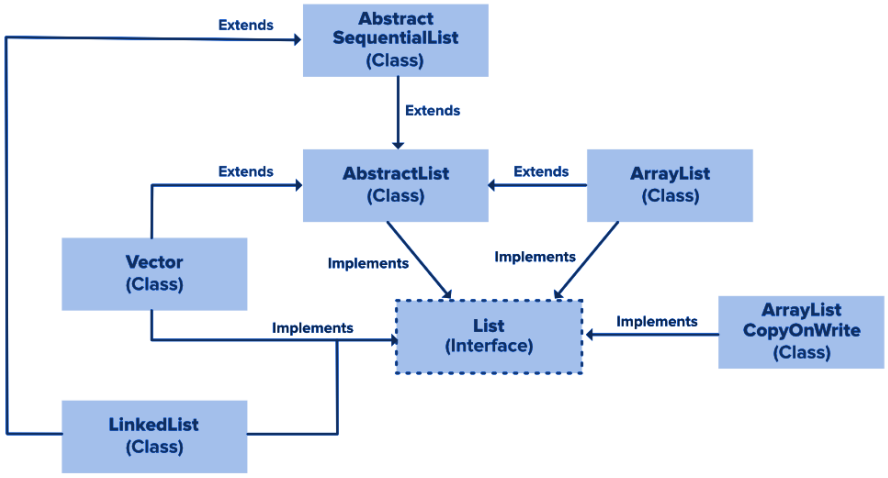
AWESOME !

Here are the Download Links and Official Documentation

Maven **(Needs to be installed in your system before next session)**
1. Download - https://maven.apache.org/install.html
How to download on windows : https://www.youtube.com/watch?v=3ODSQ0EpoQI
How to download on Mac : https://www.youtube.com/watch?v=j0OnSAP-KtU
2. Resources - https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

ArrayList in Java

ArrayList is a part of collection framework and is present in java.util package. It provides us with dynamic arrays in Java. This class is found in java.util package.



Lets Code :

Spring Chitkara University Live [SCUL] 📚⚽

Create a class . Inside the main function , Create an arraylist of type Integer .
Add 10 elements to arraylist using a loop .
Print the arraylist .
Add 100 at position index 2.
Print the arraylist .
Set 200 at position index 2.
Print the arraylist .
Remove the element at index 3
Print the arraylist .
Iterate the ArrayList and print double of each element in the arrayList .
Eg:

```
for (int num : arrli)
            System.out.print(num + " ");
```

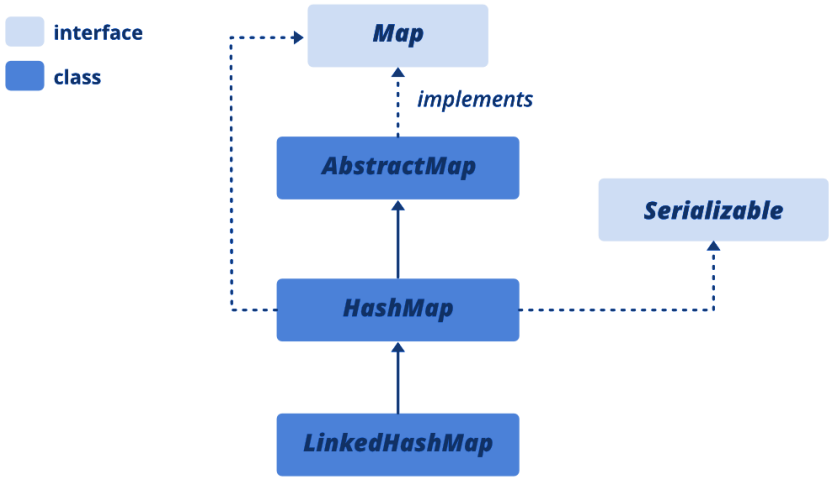
HashMap in Java with Examples

HashMap class is found in java.util package. It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs, and you can access them by an index of another type (e.g. an Integer). One object is used as a key (index) to another object (value). If you try to insert the duplicate key, it will replace the element of the corresponding key.

Lets code :
Create a HashMapExample class .
Inside main function, create a hashmap with key as String and object as Integer .
Print the map size .
Now put 4 key value pairs inside Hashmap .
Print the map size .
Print the map .
Remove one of the key-value pair using the map.remove(key) .
Print the Hashmap .
Get the value for one key in the map .
Iterate the hashmap using the for each loop .
Eg:

```
for (Map.Entry<String, Integer> e : map.entrySet())
            System.out.println("Key: " + e.getKey()
                               + " Value: " + e.getValue());
```

Spring Chitkara University Live [SCUL] 📚⚽



EXCEPTION HANDLING

Suppose you are writing a division program in java . You are taking the dividend and divisor from the user as input . If user enters 0 as divisor , then program will give ArithmeticException .
To handle this error , we have exception handling . But how will you handle it ?

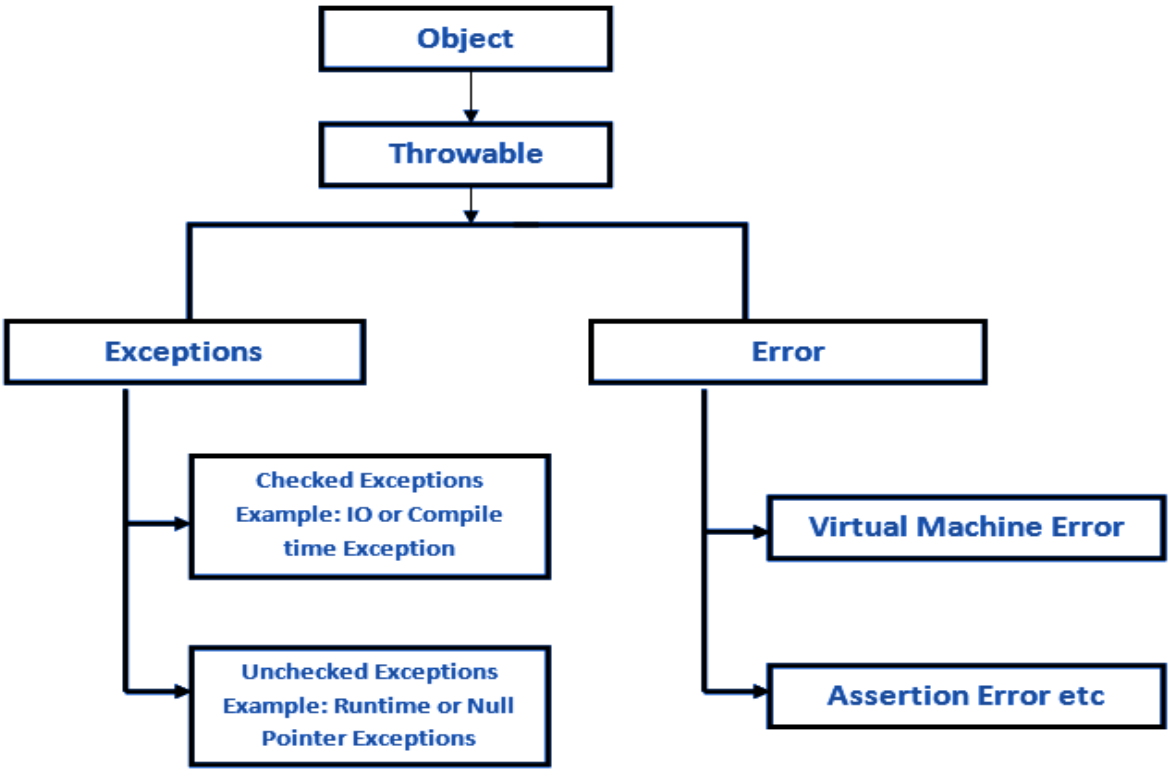
```
public static int division(int divd, int div){
    return divd/div;
}
```

WHY EXCEPTION HANDLING ?

1.Separation from Regular code (using try block where business logic is kept with a catch block with handling of possible cases)
2.Propagating the error up in the function call stack
3.Grouping and differentiating Error Conditions

Spring Chitkara University Live [SCUL] 📚⚽

Exception Hierarchy



How to use try-catch clause

```
try {
// block of code to monitor for errors
// the code you think can raise an exception
}
catch (ExceptionType1 exOb) {
// exception handler for ExceptionType1
}
catch (ExceptionType2 exOb) {
// exception handler for ExceptionType2
}
// optional
finally {
// block of code to be executed after try block ends
}
```

Spring Chitkara University Live [SCUL] 📚⚽

Checked vs Unchecked Exceptions in Java

In Java, there are two types of exceptions:

1) **Checked**: are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.

For example, consider the following Java program that opens file at location "C:\test\a.txt" and prints the first three lines of it. The program doesn't compile, because the function main() uses FileReader() and FileReader() throws a checked exception *FileNotFoundException*. It also uses readLine() and close() methods, and these methods also throw checked exception *IOException*

```
import java.io.*;

class Main {
	public static void main(String[] args) {
		FileReader file = new FileReader("C:\\test\\a.txt");
		BufferedReader fileInput = new BufferedReader(file);

		// Print first 3 lines of file "C:\test\a.txt"
		for (int counter = 0; counter < 3; counter++)
			System.out.println(fileInput.readLine());

		fileInput.close();
	}
}
```

Output:

```
Exception in thread "main" java.lang.RuntimeException:
Uncompilable source code -
unreported exception java.io.FileNotFoundException; must be
caught or declared to be
thrown
```

```
    at Main.main(Main.java:5)
```

To fix the above program, we either need to specify list of exceptions using throws, or we need to use try-catch block. We have used throws in the below program. Since FileNotFoundException is a subclass of IOException, we can just specify IOException in the throws list and make the above program compiler-error-free.

```
import java.io.*;

class Main {
	public static void main(String[] args) throws IOException {
		FileReader file = new FileReader("C:\\test\\a.txt");
		BufferedReader fileInput = new BufferedReader(file);

		// Print first 3 lines of file "C:\test\a.txt"
		for (int counter = 0; counter < 3; counter++)
			System.out.println(fileInput.readLine());

		fileInput.close();
	}
}
```

Output: First three lines of file “C:\test\a.txt”

2) **Unchecked** are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.

In Java exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under throwable is checked.

Consider the following Java program. It compiles fine, but it throws *ArithmeticException* when run. The compiler allows it to compile, because *ArithmeticException* is an unchecked exception.

Spring Chitkara University Live [SCUL] 📚⚽

```
class Main {
public static void main(String args[]) {
        int x = 0;
        int y = 10;
        int z = y/x;
}
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: /
by zero
    at Main.main(Main.java:5)
Java Result: 1
```

Exception Handling in java is managed via five keywords: try, catch, throw, throws, and finally.

Keyword	Description
try	This keyword is used to specify a block and this block must be followed by either catch or finally. That is, we can't use try block alone.
catch	This keyword must be preceded by a try block to handle the exception and can be followed by a final block later.
finally	This keyword is used to execute the program, whether an exception is handled or not.
throw	This keyword is used to throw an exception.
throws	This keyword is used to declare exceptions.

Spring Chitkara University Live [SCUL] 📚⚽

throw

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

```
// Java program that demonstrates the use of throw
class Test
{
        public static void main(String[] args)
        {
                System.out.println(1/0);
        }
}
```

throws

In a program, if there is a chance of raising an exception then compiler always warn us about it and compulsorily we should handle that checked exception, Otherwise we will get compile time error saying unreported exception XXX must be caught or declared to be thrown. To prevent this compile time error we can handle the exception in two ways:

1. By using try catch
2. **By using throws keyword**

```
// Java program to demonstrate working of throws
class ThrowsExecp
{
        static void fun() throws IllegalAccessException
        {
```

```
            System.out.println("Inside fun(). ");
            throw new IllegalAccessException("demo");
        }
        public static void main(String args[])
        {
            try
            {
                fun();
            }
            catch(IllegalAccessException e)
            {
                System.out.println("caught in main.");
            }
        }
}
```

Lets recap :

Important points to remember about throws keyword:

- throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
- By the help of throws keyword we can provide information to the caller of the method about the exception

Note : All code files , notes and download links have been uploaded to Github .
Please make sure to download Java in your systems , IntelliJ as IDE and Maven in your systems as build automation tool .