Exception handling

TryOut: Exception

```
class NewClass
{
        public static void main(String[] args)
        {
                x();
                }
        static void x()
        {
                 y();
                 }
        static void y()
        {
                 int a = 7/0;
                 }
}
```

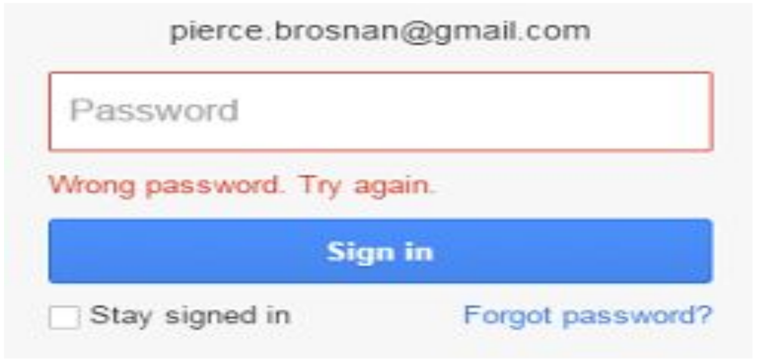
```
class NewClass
{
        public static void main(String[] args)
        {
                int sum = 0;
                int totalSubjects = 0;
                for (int i = 0; i <= marks.length; i++) {
                        sum += marks[i];
                }
     this.averageMarks = sum / totalSubjects;
     System.out.println("Average Marks : " + this.averageMarks);

                 }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at NewClass.main(NewClass.java:20)
```

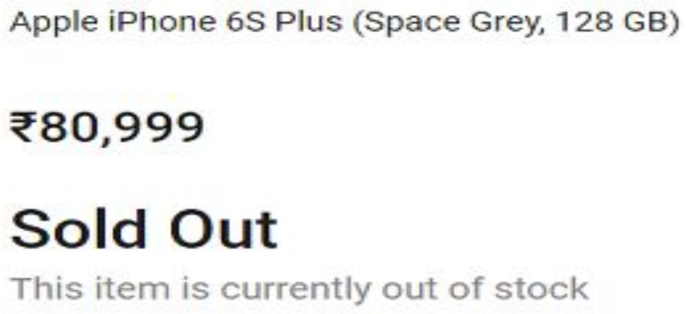
Real Life scenario:

1. "Invalid username or password"


pierce.brosnan@gmail.com

Wrong password. Try again.

2. "Out of stock!"

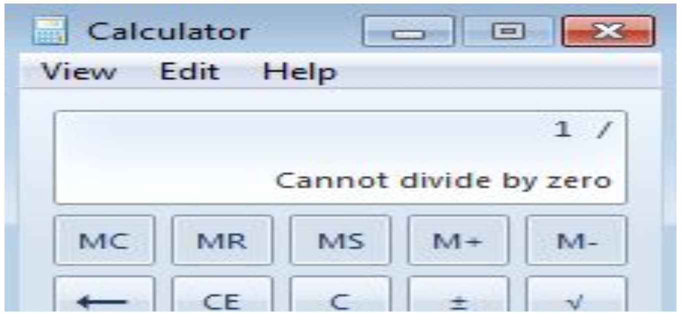

Apple iPhone 6S Plus (Space Grey, 128 GB)

₹80,999

Sold Out

This item is currently out of stock

3. "Cannot divide by zero"



```
1 /
Cannot divide by zero
```

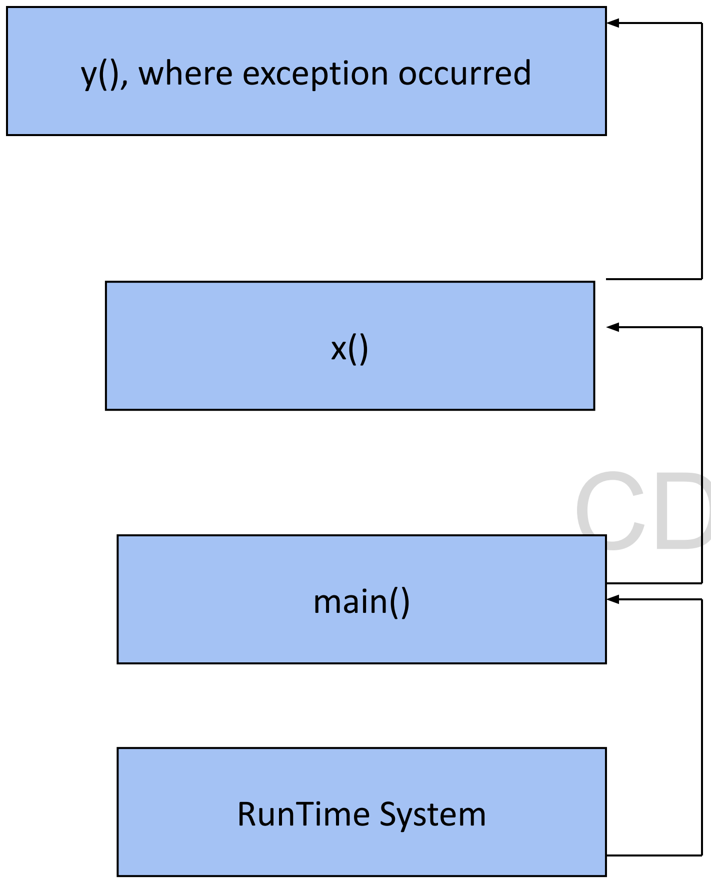
What is an Exception?

- Exception is an event which disrupts the normal flow of program during the program's execution. In an application, usually a certain input, or a programming mistake or an overlook can lead to exceptions.
- They come out as runtime errors and abnormally terminate the program.
- Process of handling the exceptions is known as Exception Handling.
- It would be better if user-friendly and meaningful error messages can be shown to the end user.

For example:-

```
class NewClass{
      public static void main(String[] args){
         System.out.println("Main  Starts");
         System.out.println(5/0);
         System.out.println("Main ends");
         // do something else
      }
}
```

Understanding the Call Stack



```
class NewClass
{
        public static void
    main(String[] args)
        {
                x();
                }
        static void x()
        {
                y();
                }
        static void y()
        {
                int a = 7/0;
                }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at NewClass.y(NewClass.java:23)
        at NewClass.x(NewClass.java:21)
        at NewClass.main(NewClass.java:19)
```

Where Exception may Occur?

Exceptions can occur at many levels:

- **Hardware/operating system level.**
  - Arithmetic exceptions; divide by 0, under/overflow.
  - Memory access violations, stack over/underflow.
- **Language level.**
  - Type conversion: illegal values, improper casts.
  - Bounds violations: illegal array indices.
  - Bad references: null pointers.
- **Program level.**
  - User defined exceptions.

Exception Handler

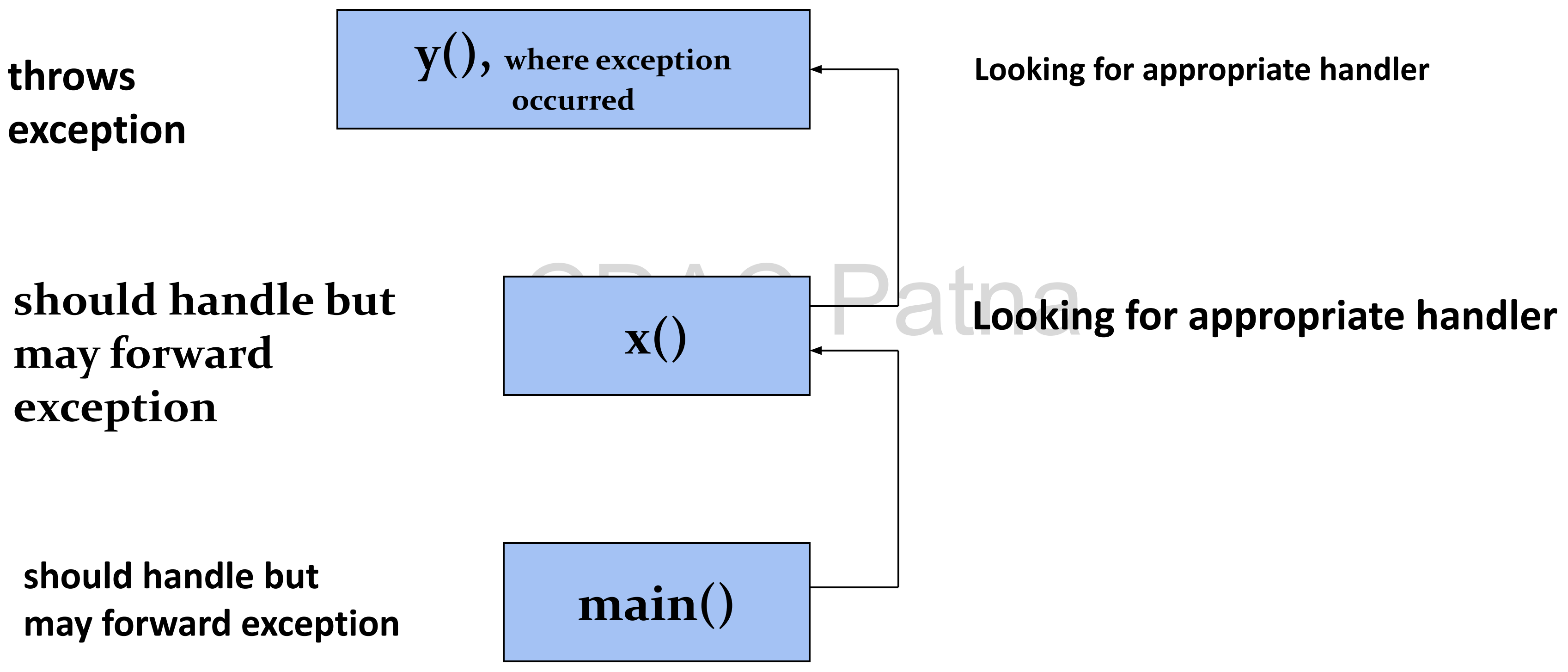
- The runtime system searches the call stack for a method that contains a block of code that can handle the exception. This block of code is called an exception handler.

- The search begins with the method in which the error occurred and proceeds through the call stack in the reverse order in which the methods were called.

- When an appropriate handler is found, the runtime system passes the exception object to the handler.

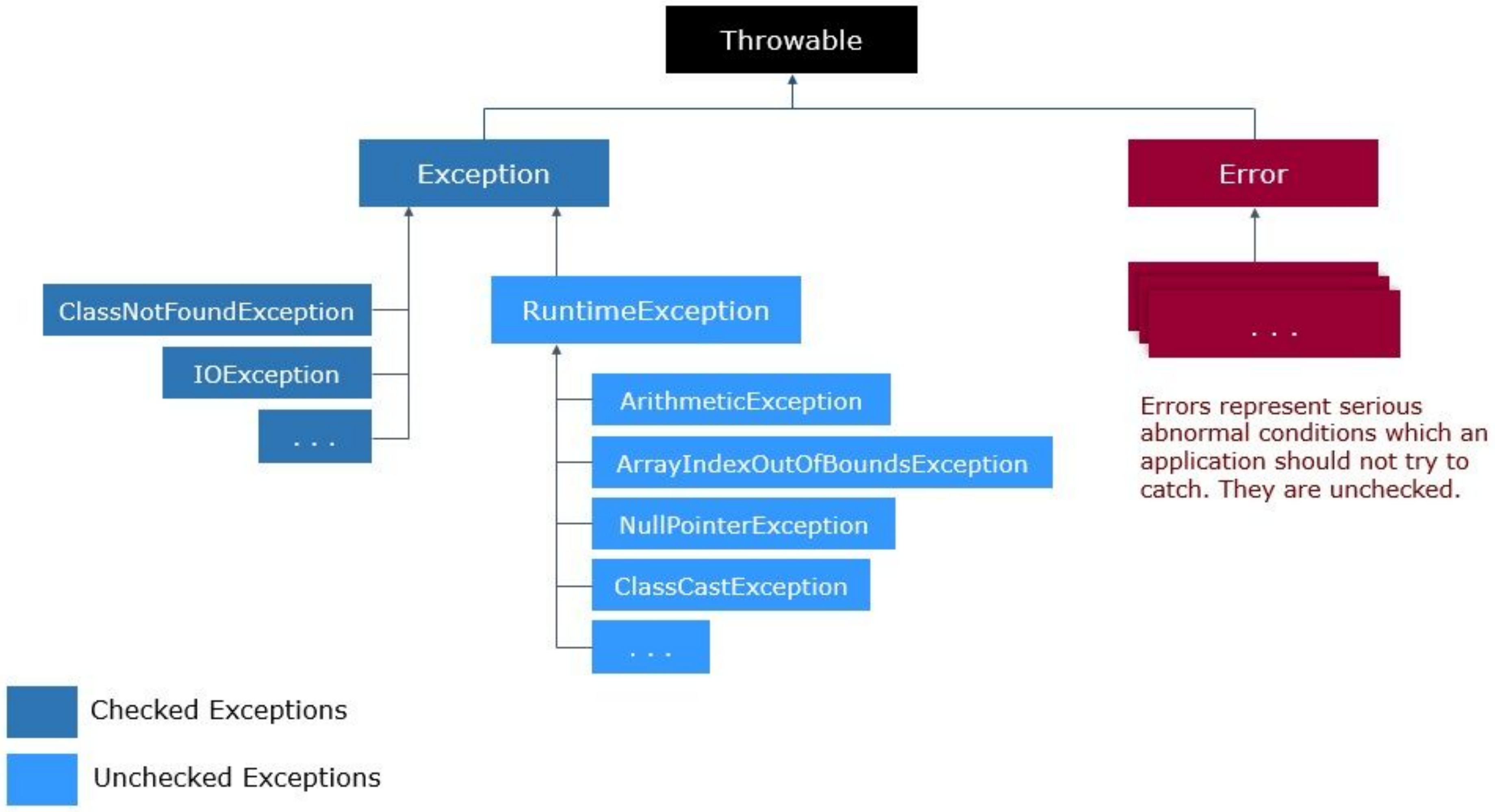
- An exception handler is considered appropriate if the type of the exception object thrown matches the type that can be handled by the handler.

Searching for the Exception Handler in the Call Stack

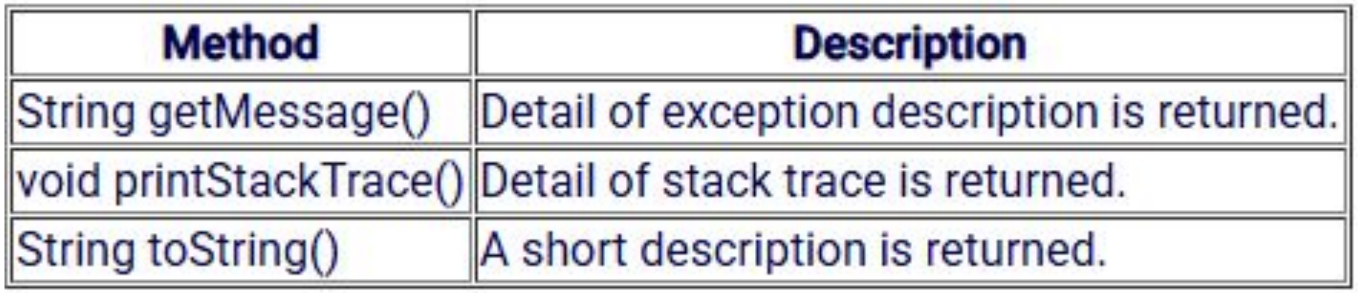

throws exception
y(), where exception occurred
Looking for appropriate handler
should handle but may forward exception
x()
Looking for appropriate handler
should handle but may forward exception
main()

If main forwarded the exception, the thread will abnormally terminate.

Exception Hierarchy



Throwable class methods



Method	Description
String getMessage()	Detail of exception description is returned.
void printStackTrace()	Detail of stack trace is returned.
String toString()	A short description is returned.

Exception Types

Checked Exceptions:

- If not handled by the programmer, these exceptions will be detected during the compilation of the program which will result in compilation errors.
- Programmers are forced to handle these exceptions or declare its propagation to the calling environment.

Unchecked Exceptions:

- These exceptions are detected during the execution of the program or the runtime, hence causing an error.
- Programmers are neither forced to handle it nor declare its propagation.

Runtime Exception due to Logical Errors

```
public class NewClass
{
    public static void main(String[] args)
      {
         int array[] = {6,7,8,9,10};
         for(int index=0; index<array.length+2; index++)
         System.out.print(array[index]+” “);
         System.out.println("After loop");
      }
}
```

Output:-

```
6 7 8 9 10
Exception in thread "main"java.lang.ArrayIndexOutOfBoundsException: 5 at
    NewClass.main(NewClass.java:22)
```

Runtime Exception contd.

```
public class NewClass
{
        static Integer i;
        public static void main(String[] args)
        {
                System.out.println("1st statement");
                System.out.println(i.intValue());
                System.out.println("3rd Statement");
                show("tom");
        }
        static void show(String msg)
        {
            System.out.println(msg.toUpperCase());
        }
}
```

Output:-

```
1st statement
Exception in thread "main" java.lang.NullPointerException at
      NewClass.main(NewClass.java:21)
```

Runtime Exception contd.

```
class Test {
        public static void main(String[] args) {
                System.out.println(“First statement”);
                m(“india”);
                m(35);
                System.out.println(“Last statement”);
        }
        static void m(Object o) {
                Integer i = (Integer)o;
        }
}
```

```
First statement
Exception in thread "main" java.lang.ClassCastException: java.lang.String
        at Test.m(NewClass.java:27)
        at Test.main(NewClass.java:21)
```

Java Exception Handling Keywords

- try
- catch
- finally
- throw
- throws

Handling Exceptions: try - catch block

```
try {
    // Code that can throw exceptions
}
catch(Exception1 exception1) {
    // Code for handling Exception1
}
catch(Exception2 exception2) {
    // Code for handling Exception2
}
```

TryOut: try-catch

try-catch block contd.

- Whenever the statements in the try block throw an exception, it is immediately caught by the first matching catch block which can handle it. The code inside the try block following the line causing the exception is ignored.
- The exception will remain unhandled and will be propagated to the calling function if a matching catch block is not found.
- If no exception is thrown inside a try block, the catch blocks following it are ignored.
- A catch block that can handle objects of Exception class can catch all the exceptions. This should always be the last catch block in the catch sequence.

Handling Exceptions: Finally

```
try {
    // Code that can throw exceptions
}
catch(Exception1 exception1) {
}
finally {
    // Code to be executed no matter what
}
```

tryout: finally

Handling Exceptions: throw Keyword(checked)

- Till now the the Java program created an threw any exception on its own. But now, we have to handle the reins of the same.
- Java allows us to explicitly generate or throw exceptions using the throw keyword:

```
Exception e = new Exception(<<message in String format>>);
throw e;
```

- Any object of type Throwable can be thrown.
- Since checked exceptions are detected during compiletime, handling of it is forced on the programmer.

Tryout: throw keyword

Handling Exceptions: throws Keyword(checked)

- We have been handling exceptions in the method in which they are thrown. If we need to propagate and handle the exceptions elsewhere then we will use **throws.**
- If there is a checked exception which the method doesn't handle, it has to be declared using the **throws** clause:
- One of the main purpose of handling the exception was so that the end user is not shown the exception stack trace.

tryout: throws

Handling Exceptions(Unchecked)

- Since unchecked exceptions are detected during runtime, handling of it is not forced on the programmer.
- So Runtime System automatically throws this type of exceptions to calling method.
- There is no need to explicitly mention throws keyword in methods.

Tryout: Unchecked exception

User - Defined Exceptions

- A user-defined exception is any class that is a subclass of the Exception class. In other terms, if we extend the Exception class, we get our user-defined exception.

```
public class UserDefinedException extends Exception {}

public class StockNotAvailableException extends Exception {
public UserDefinedException (String message){
super(message);
}
}
```

- We are extending the main Exception class thereby making our exception a checked exception. A user-defined exception can also be made an unchecked exception by extending the RuntimeException class or any of its subclasses.

Tryout : User - Defined Exceptions

Assignments

1. Implement a function that validates a user's age and throws an IllegalArgumentException if the age is less than 18 using the throw keyword.
2. Write a Java program that handles both ArithmeticException and ArrayIndexOutOfBoundsException using a multi-catch block.
3. Implement a method that throws a custom unchecked exception EmptyStringException when the input string is empty or null.
4. Develop a program that demonstrates the behavior of the finally block when an exception is thrown and caught inside it.
5. Write a Java program that throws a custom unchecked exception NegativeSalaryException if an employee's salary is entered as a negative value.
6. Create a program that takes an array of 5 integers from the user and tries to access the 10th index, handling the exception gracefully.(ArrayIndexOutOfBoundsException).

Thanks