Java Exception Handling

BY: SAGAR PATEL

Content

- What is Exception?
- What is Exception Handling?
- Advantages of Exception Handling.
- Hierarchy of exception handling.
- Types of Exception.
- Difference between checked and unchecked exceptions.
- Common scenarios.
- Java Exception Handling Keywords.

Introduction

- The exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

What is Exception?

- Dictionary Meaning: **Exception is an abnormal condition**.
- In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is exception handling?

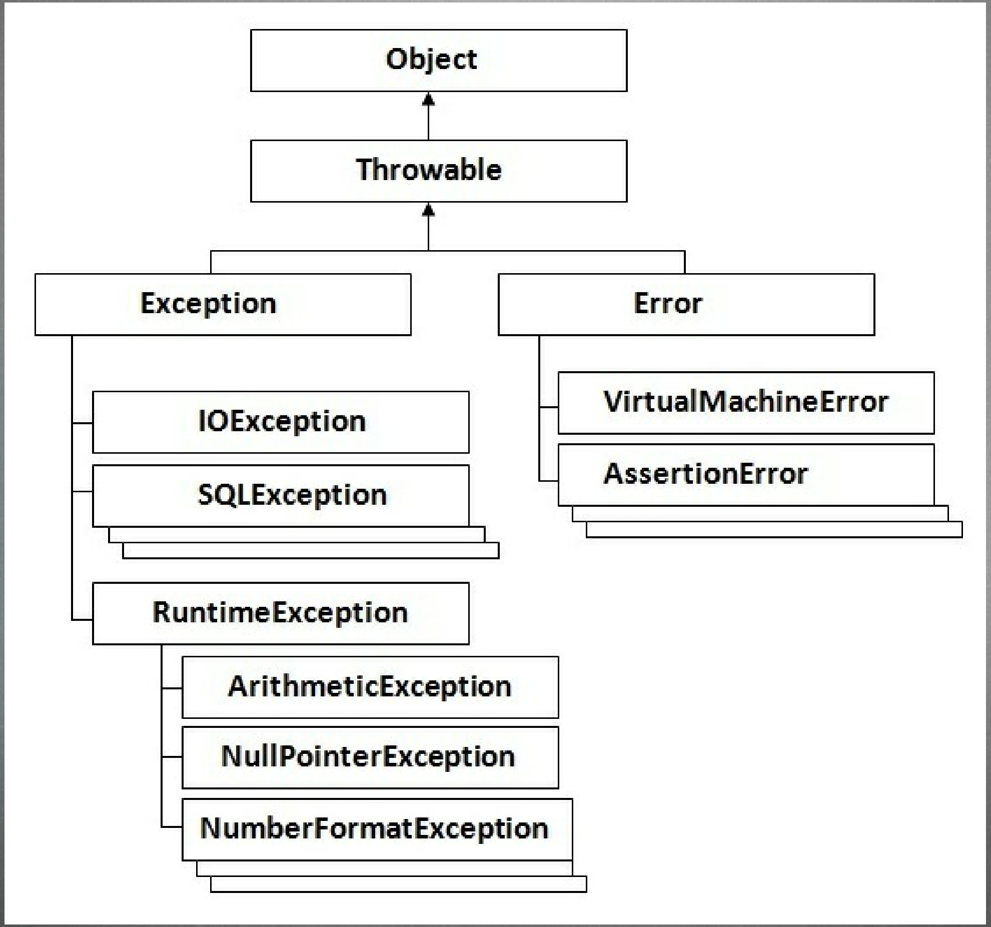
- Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

Advantage of Exception Handling

- The core advantage of exception handling is to maintain the normal flow of the application.
- Exception normally disrupts the normal flow of the application that is why we use exception handling.
- Let's take a scenario:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;  //exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Hierarchy of exception handling.



Types of Exception.

- There are mainly two types of exceptions:
  - checked and unchecked
- where error is considered as unchecked exception.
- The sun microsystem says there are three types of exceptions:
  - Checked Exception
  - Unchecked Exception
  - Error

Difference between checked and unchecked exceptions.

	Definition	Example	Runtime/ Compile Time
Checked Exception	The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions.	IOException, SQLException etc.	Checked exceptions are checked at compile-time.
Unchecked Exception	The classes that extend RuntimeException are known as unchecked exceptions.	ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.	Unchecked exceptions are checked at runtime.
Error	Error is irrecoverable.	OutOfMemoryError, VirtualMachineError, AssertionError etc.	-

Common scenarios.

- There are given some scenarios where unchecked exceptions can occur. They are as follows:

1. Scenario where ArithmeticException occurs…
   - If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;                    //ArithmeticException
```

2. Scenario where NullPointerException occurs
   - If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
String s=null;
System.out.println(s.length());      //NullPointerException
```

Common scenarios. (cont…)

3. Scenario where NumberFormatException occurs.

   - The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

```
String s="abc";
int i=Integer.parseInt(s);                    //NumberFormatException
```

4. Scenario where ArrayIndexOutOfBoundsException occurs.

   - If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

```
int a[]=new int[5];
a[10]=50;                              //ArrayIndexOutOfBoundsException
```

Java Exception Handling Keywords.

- There are 5 keywords used in java exception handling.
  1. try
  2. catch
  3. finally
  4. throw
  5. throws

Java try-catch

- Java try block is used to enclose the code that might throw an exception. It must be used within the method.
- Java try block must be followed by either catch or finally block.
- **Syntax of java try-catch**

```
try{
    //code that may throw exception
}catch(Exception_class_Name ref){}
```

- **Syntax of try-finally block**

```
try{
    //code that may throw exception
}finally{}
```

Problem without exception handling

```
public class Testtrycatch1{
  public static void main(String args[]){
      int data=50/0;          //may throw exception
      System.out.println("rest of the code...");
  }
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
```

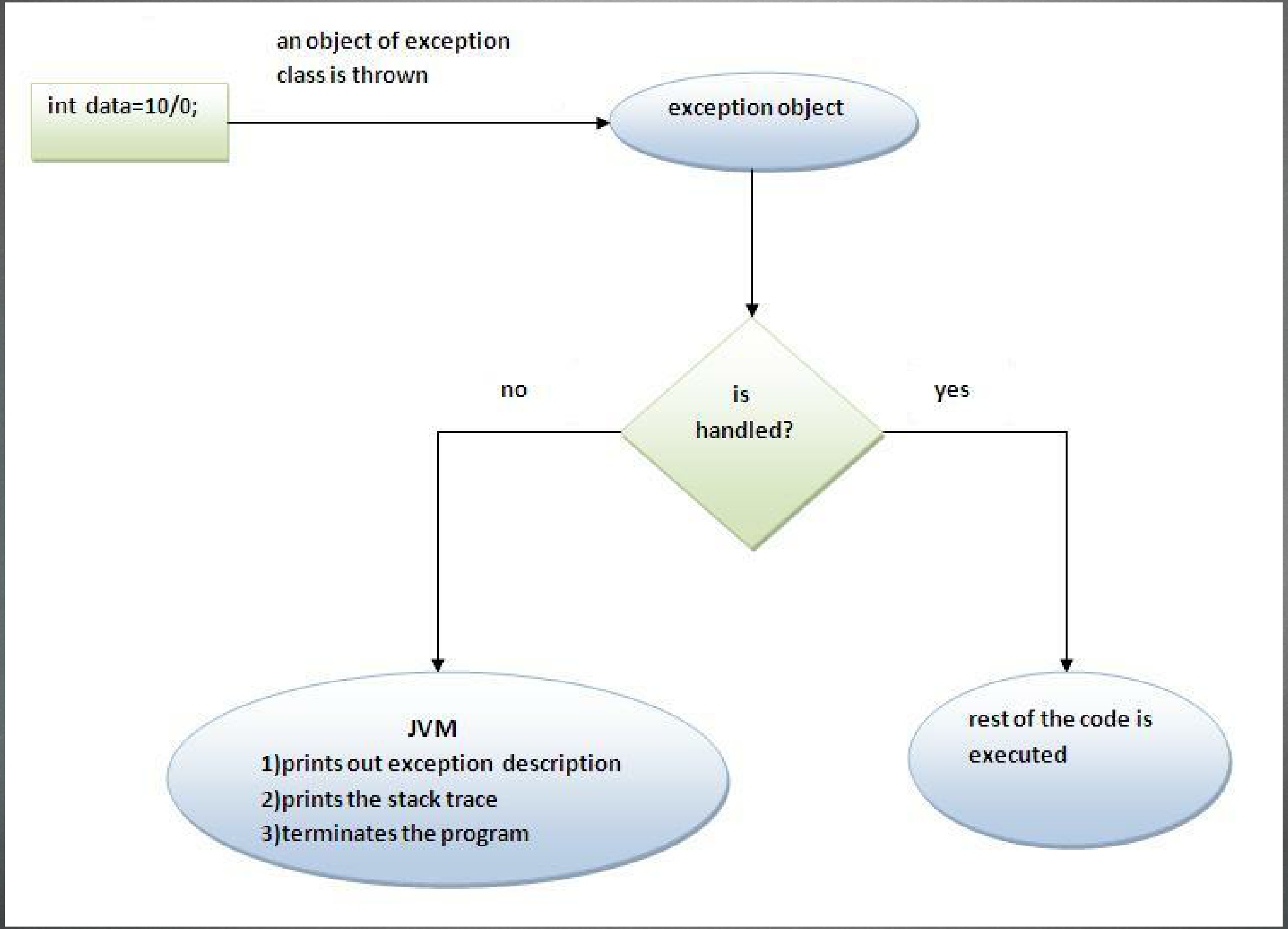
Solution by exception handling

```
public class Testtrycatch2{
  public static void main(String args[]){
    try{
            int data=50/0;
    }catch(ArithmeticException e){System.out.println(e);}
    System.out.println("rest of the code...");
  }
}
```

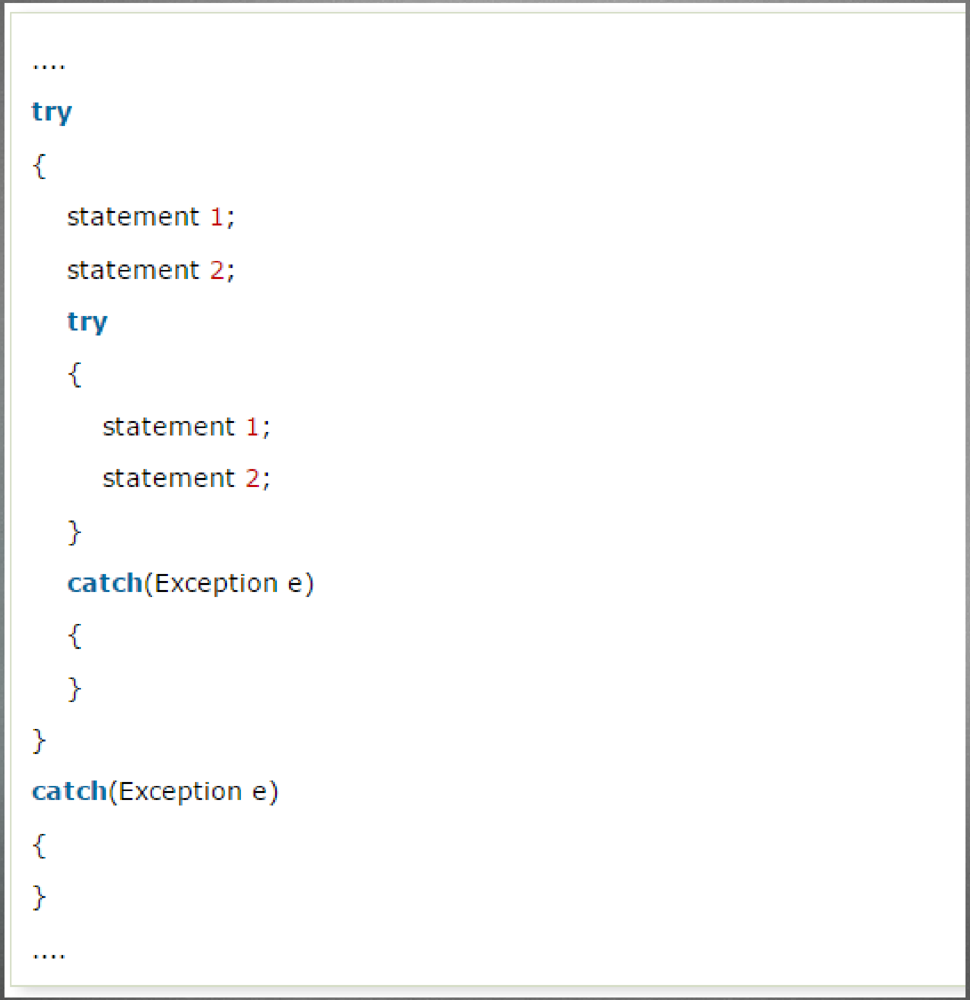
Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Java try-catch

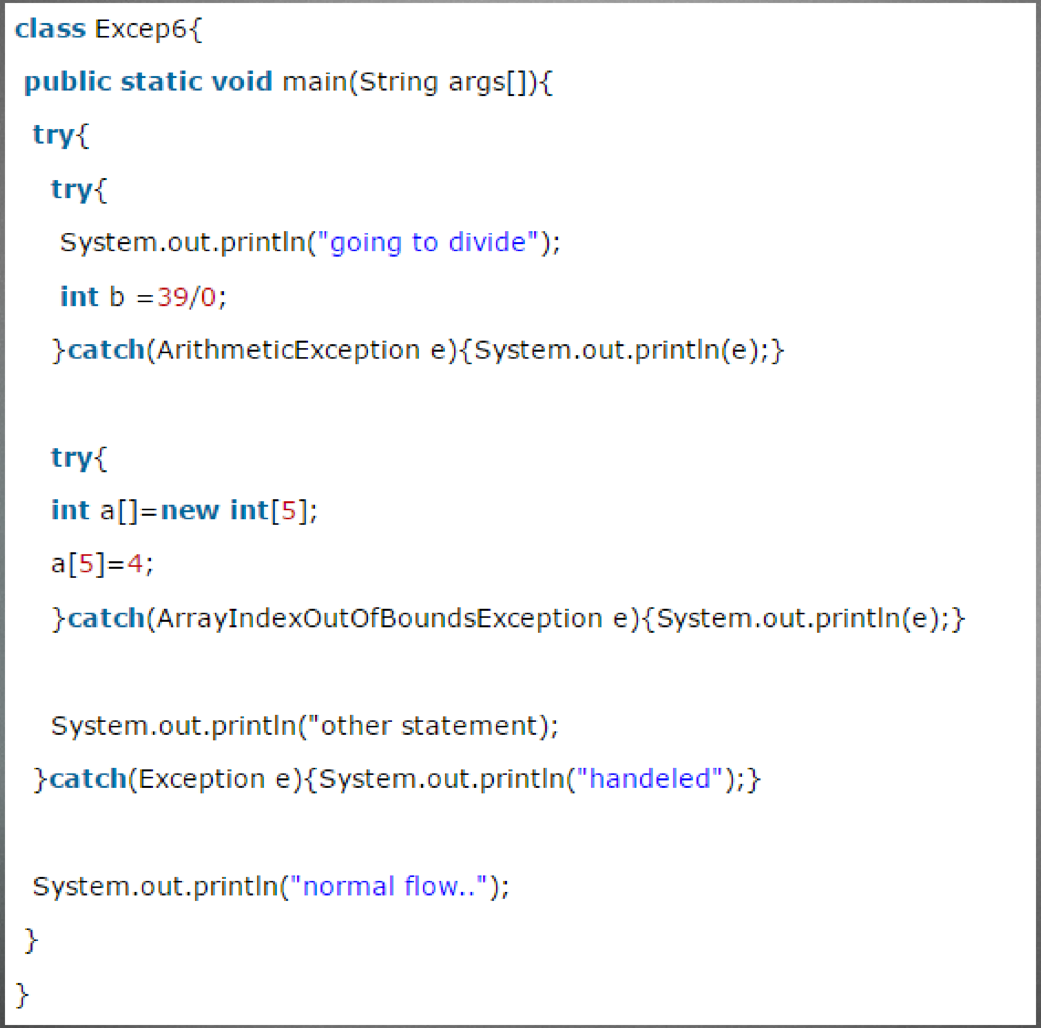

an object of exception class is thrown
int data=10/0;
exception object
is handled?
no
yes
JVM
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

Nested try catch



```
....
try
{
    statement 1;
    statement 2;
    try
    {
        statement 1;
        statement 2;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
{
}
....
```

Nested try catch



```
class Excep6{
 public static void main(String args[]){
  try{
    try{
     System.out.println("going to divide");
     int b =39/0;
    }catch(ArithmeticException e){System.out.println(e);}

    try{
    int a[]=new int[5];
    a[5]=4;
    }catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

    System.out.println("other statement);
  }catch(Exception e){System.out.println("handeled");}

  System.out.println("normal flow..");
 }
}
```

Java throw exception

- The Java throw keyword is used to **explicitly throw an exception**.
- We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.
- The syntax of java throw keyword is given below.
  - **throw** exception;
- **Example,**
  - **throw new** IOException("sorry device error);

Java throws keyword

- The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
- Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Java throws keyword (Cont…)

- Syntax of java throws

```
return_type method_name() throws exception_class_name{
   //method code
}
```

- Which exception should be declared?
  - checked exception only, because:
  - **unchecked Exception:** under your control so correct your code.
  - **error:** beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

Java throws keyword (Cont…)

- **Rule: If you are calling a method that declares an exception, you must either caught or declare the exception.**

- There are two cases:
- Case1:You caught the exception i.e. handle the exception using try/catch.
- Case2:You declare the exception i.e. specifying throws with the method.

throw vs throws

#	Throw	Throws
1.	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2.	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3.	Throw is followed by an instance.	Throws is followed by class.
4.	Throw is used within the method.	Throws is used with the method signature.
5.	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException, SQLException.

Difference between final, finally and finalize

Final	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Final is a keyword.
Finally	Finally is used to place important code, it will be executed whether exception is handled or not.	Finally is a block.
Finalize	Finalize is used to perform clean up processing just before object is garbage collected.	Finalize is a method.

Java final example

```
class FinalExample{
  public static void main(String[] args){
    final int x=100;
    x=200;        //Compile Time Error
  }
}
```

Java finally example

```
class FinallyExample{
  public static void main(String[] args){
    try{
      int x=300;
    }catch(Exception e){System.out.println(e);}
      finally{System.out.println("finally block is
                  executed");}
  }
}
```

Java finalize example

```
class FinalizeExample{
  public void finalize(){
    System.out.println("finalize called");}
  public static void main(String[] args){
    FinalizeExample f1=new FinalizeExample();
    FinalizeExample f2=new FinalizeExample();
    f1=null;
    f2=null;
    System.gc();
  }
}
```

Exception Handling with Method Overriding in Java

1. If the superclass method does not declare an exception
   - If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.
2. If the superclass method declares an exception
   - **Rule**: If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

1. If the superclass method does not declare an exception

```
class A{
  public void show(){
    System.out.println(“Hello”);
  }
}
class B extends A{
  public void show() throws IOException {
    System.out.println(“Hi”);
  }
}
```

*InValid

1. If the superclass method does not declare an exception

```
class A{
  public void show(){
    System.out.println("Hello");
  }
}
class B extends A{
  public void show() throws ArithmeticException {
    System.out.println("Hi");
  }
}
```

*Valid

2. If the superclass method declares an exception

```
class A{
   public void show() throws ArithmeticException{
      System.out.println("Hello");
   }
}
class B extends A{
   public void show() throws Exception {
      System.out.println("Hi");
   }
}
```

*InValid

2. If the superclass method declares an exception

```
class A{
   public void show() throws Exception{
      System.out.println("Hello");
   }
}
class B extends A{
   public void show() throws Exception {
      System.out.println("Hi");
   }
}
```

*Valid

2. If the superclass method declares an exception

```
class A{
   public void show() throws Exception{
      System.out.println(“Hello”);
   }
}
class B extends A{
   public void show() throws ArithmeticException {
      System.out.println(“Hi”);
   }
}
```

*Valid

2. If the superclass method declares an exception

```
class A{
   public void show() throws Exception{
      System.out.println(“Hello”);
   }
}
class B extends A{
   public void show() {
      System.out.println(“Hi”);
   }
}
```

*Valid

Java Custom Exception

1. Create custom exception

```
class InvalidAgeException extends Exception{
  InvalidAgeException(String s){
    super(s);
  }
}
```

Create Class which use custom exception

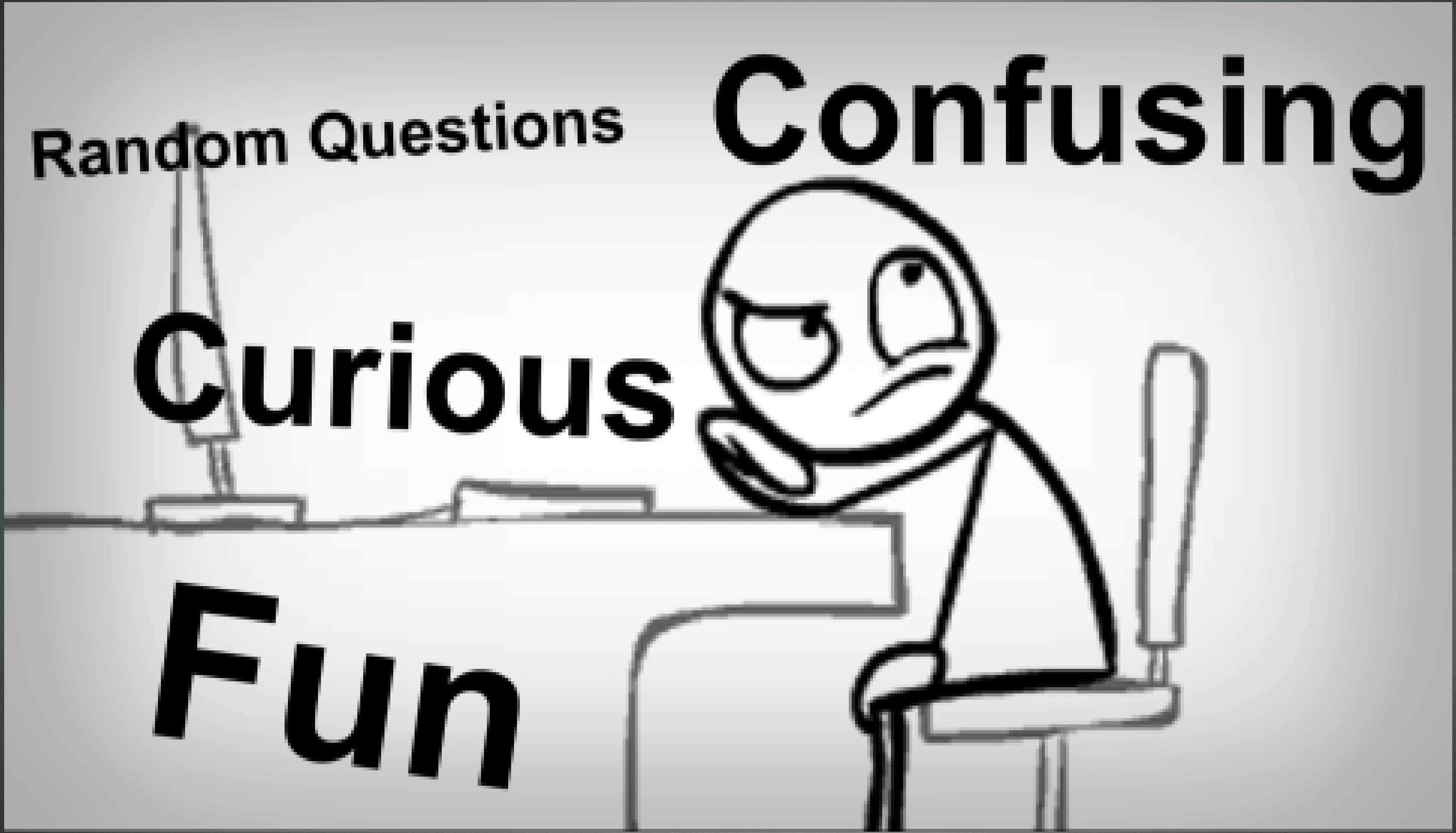
```
class TestCustomException1{

   static void validate(int age)throws InvalidAgeException{
     if(age<18)
      throw new InvalidAgeException("not valid");
     else
      System.out.println("welcome to vote");
   }
   public static void main(String args[]){
      try{
      validate(13);
      }catch(Exception m){System.out.println("Exception occured: "+m);}

      System.out.println("rest of the code...");
  }
}
```

Output:

```
Exception occurred: InvalidAgeException: not valid
rest of the code...
```

Referances

- http://www.javatpoint.com/exception-handling-in-java
- http://www.tutorialspoint.com/java/java_exceptions.htm


Random Questions
Confusing
Curious
Fun

Thanks