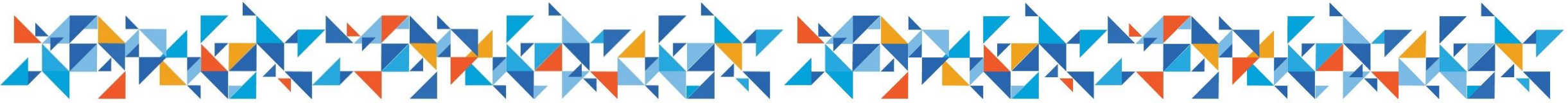
پوهنتون کاردان
KARDAN UNIVERSITY

Object Oriented Programming (JAVA)

Exception Handling

Learning Outcomes

- What is an exception?
- Types of Exceptions
- Checked Exceptions
- Unchecked Exceptions
- Finally block
- Throw Statement



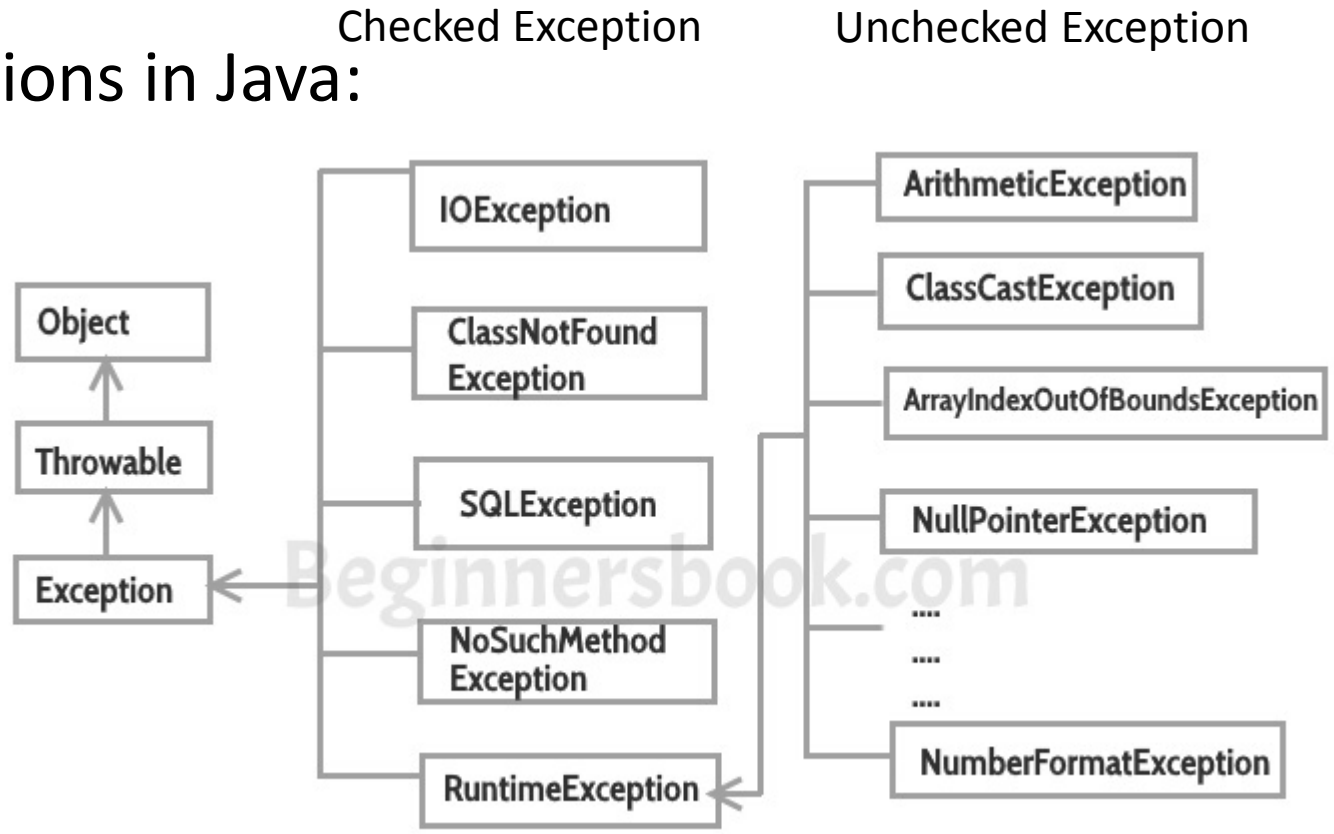
What is an Exception

- An exception is an indication of a problem that occurs during a program's execution.
- Exception is unwanted situation of program
- Exception handling enables you to create applications that can resolve (or handle) exceptions.

- It will help you write *robust* and *fault-tolerant* programs that can deal with problems and continue executing or *terminate gracefully*.

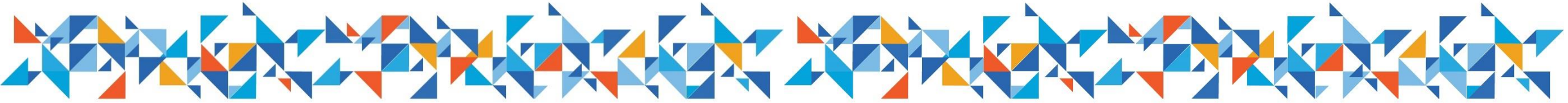
Types of Exception

- There are two types of exceptions in Java:
  1)Checked exceptions
  2)Unchecked exceptions



Checked / Compile time exceptions

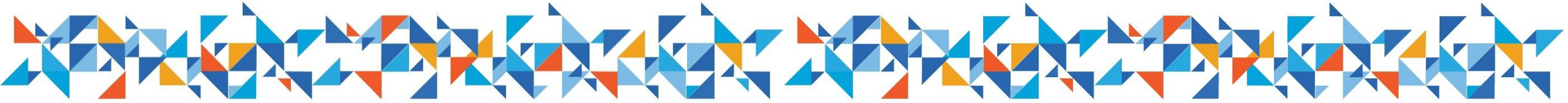
- Generally, checked exceptions denote error scenarios which are outside the immediate control of the program. They occur usually interacting with outside resources/network resources e.g. database problems, network connection errors, missing files etc.
- All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation



Unchecked /Runtime Exceptions

- Java also provides UncheckedExceptions, the occurrences of which are **not checked by the compiler**. They will come into life/occur into your program, once any buggy code is executed.
- Runtime Exceptions are known as Unchecked Exceptions. These exceptions are not checked at compile-time
- It’s the responsibility of the programmer to handle these exceptions and provide a safe exit.

- For example: ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc



Example: Divide by Zero without Exception

```
 2 import java.util.Scanner;
 3 public class Zero {
 4     public static void main (String ar[]){
 5
 6         Scanner scan=new Scanner(System.in);
 7
 8         System.out.println("Enter the Numerator: ");
 9         int a=scan.nextInt();
10         System.out.println("Enter the Denominator: ");
11         int b=scan.nextInt();
12
13         System.out.println("The Division result is: "+a/b);
14     }
```

Console

<terminated> Zero [Java Application] C:\Program Files\Java\jre1.8.0_151\bin\javaw.exe (Jan 5, 2019, 11:19:52 PM)

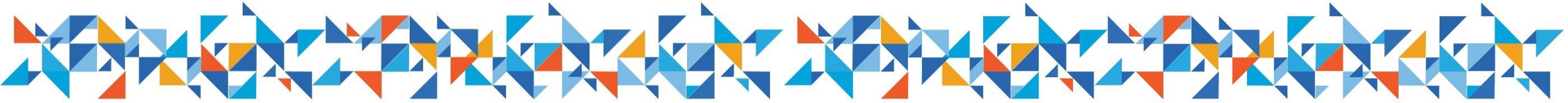
```
Enter the Numerator:
12
Enter the Denominator:
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at test.Zero.main(Zero.java:14)
```

Stack trace

- Several lines of information are displayed in response to this invalid input. This information is known as **stack trace**.

- The stack trace includes the path of execution that led to the exception method by method. This helps you debug the program.

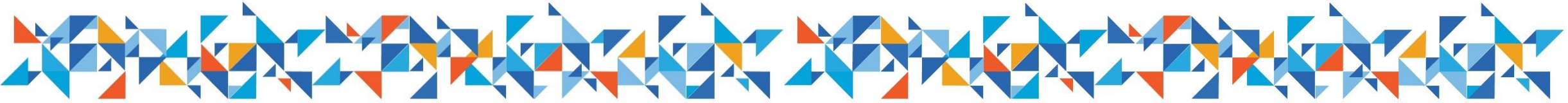
```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at test.Zero.main(Zero.java:14)
```



ArithmeticException

- ***The exception which is generated when a number is divided by zero is called ArithmeticException.***
- ***Use try & catch to solve the above exception***

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at test.Zero.main(Zero.java:14)
```



Syntax

Handle exceptions by enclosing risky code in a “try” block and catching exceptions with a “catch” block.

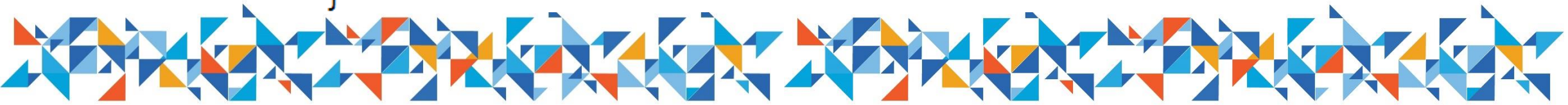
```
try {
  //  Block of code to try
}
catch(Exception e) {
  //  Block of code to handle errors
}
```

Handling an Arithmetic Exception

```
import java.util.Scanner;
public class NewClass {
    public static void main(String[] args) {

        Scanner obj=new Scanner(System.in);
        try{
        System.out.println("Enter numerator");
        int num1=obj.nextInt();
        System.out.println("Enter denominator");
        int num2=obj.nextInt();

        int div = num1 / num2;
        System.out.println("The division result is: "+div);
        }catch(ArithmeticException e){
            System.err.println("Number cannot be divided by zero"+e);
        }
    }}
```

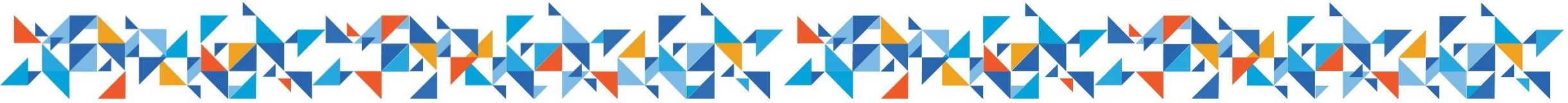
Handling an Arithmetic Exception


```
public static void main (String ar[]){

    Scanner scan=new Scanner(System.in);

    boolean test=true;
    while(test){
    try{
    System.out.println("Enter the Numerator: ");
    int a=scan.nextInt();
    System.out.println("Enter the Denominator: ");
    int b=scan.nextInt();

    System.out.println("The Division result is: "+a/b);
    test=false;
    }
    catch(ArithmeticException art){
        System.err.println("Can't Divide by Zero! "+art);
    }
    }
```



Contd..

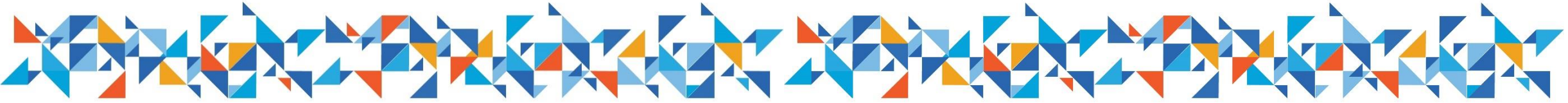
```
Enter the Numerator:
12
Enter the Denominator:
0
Enter the Numerator:
Can't Divide by Zero! java.lang.ArithmeticException: / by zero
22
Enter the Denominator:
2
The Division result is: 11
```



Input Mismatch Exception

- Scanner nextInt() is expecting number instead of a String.

```
Enter the Numerator:
12
Enter the Denominator:
Goal
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at test.Zero.main(Zero.java:11)
```

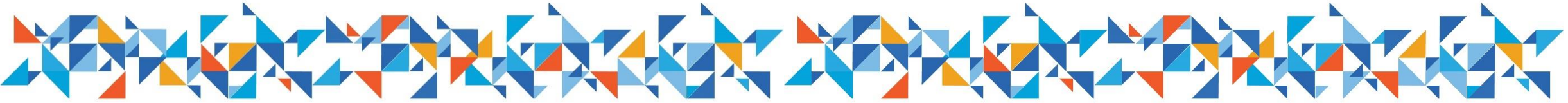


Handling input Mismatch Exception

```
import java.util.InputMismatchException;
import java.util.Scanner;
public class inputmismatchexception {
        public static void main(String[] args) {
        Scanner input=new Scanner(System.in);
        boolean test=true;
        do{
        try {
        System.out.println("Enter numerator");
        int num1=input.nextInt();
        System.out.println("Enter Denominator");
        int num2=input.nextInt();

        int div=num1/num2;
        System.out.println(div);
        test=false;
        }
        catch(InputMismatchException t){
            System.err.println("Number cannot be divided by text"+t);
            input.nextLine();
        }
        }while(test);
    }
}
```

Contd..

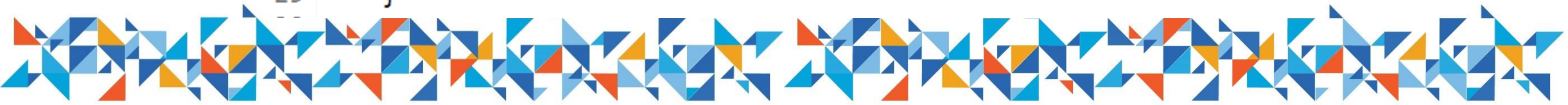
```
Enter the Numerator:
78
Enter the Denominator:
k
Please enter a number! java.util.InputMismatchException
Enter the Numerator:
55
Enter the Denominator:
5
The Division result is: 11
```



Handling Multiple exceptions

پوهنتون کاردان
KARDAN UNIVERSITY

```
11          do{
12          try{
13          System.out.println("Enter the Numerator: ");
14           a=scan.nextInt();
15          System.out.println("Enter the Denominator: ");
16           b=scan.nextInt();
17
18          System.out.println("The Division result is: "+a/b);
19          test=false;
20          }
21          catch(ArithmeticException arit){
22              System.out.println("Arithmatic Exception: "+arit);
23          }
24          catch(InputMismatchException input){
25              System.err.println("Please enter a number! "+input);
26          scan.nextLine();
27          }
28          } while(test);
29      }
```



Catching Multiple Exceptions

- By using the “ | “ sign you can place different Exceptions in one Catch

```
do{
try{
System.out.println("Enter the Numerator: ");
 a=scan.nextInt();
System.out.println("Enter the Denominator: ");
 b=scan.nextInt();

System.out.println("The Division result is: "+a/b);
test=false;
}
catch(ArithmeticException | InputMismatchException arit){
    System.out.println(" Exception: "+arit);
    scan.nextLine();
}
} while(test);
```

Finally Block

- A **finally block** contains all the crucial statements that must be executed whether exception occurs or not.

```
public class Finallyblock {
    public static void main(String[] args) {
        try{
        int num=200/0;
        System.out.println(num);
        }
        finally {
            System.out.println("This is finally block");
            String stdname="Tahir";
            int age=25;
            System.out.println("Student name is: "+stdname);
            System.out.println("Age is: "+age);
            System.out.println("****************************");
        }
        System.out.println("This is outside of finally block and try & catch ");
    }
}
```

The throw keyword

- The throw statement allows you to create a custom error.

- The throw statement is used together with an exception type.
- There are many exception types available in Java: ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc:

```java
import java.util.Scanner;
public class Throwkeyword {
static void checkage(int age){
    if(age<18){
        throw new ArithmeticException("You are not eligible to vote");
    }
    else{
        System.out.println("You are eligible to vote");
    }
}
     public static void main(String[] args) {
         Scanner obj=new Scanner(System.in);
         System.out.println("Enter age of the voter");
         int age=obj.nextInt();
    checkage(age);
}
}
```

پوهنتون کاردان
KARDAN UNIVERSITY

Thank You…!